(12) SR

IDA PAPER P-1318

LEVEL II

A COMPUTER PROGRAM FOR SOLVING SEPARABLE NONCONVEX OPTIMIZATION PROBLEMS

Jeffrey H. Grotte

with appendices by

James E. Falk
Paul F. McCoy

January 1978

INSTITUTE FOR DEFENSE ANALYSES
PROGRAM ANALYSIS DIVISION

IDA Log No. HQ 77-19991

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER P-1318	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A COMPUTER PROGRAM FOR SOLVING SEPARABLE NONCONVEX OPTIMIZATION PROBLEMS		5. TYPE OF REPORT & PERIOD COVERED Final
		6. PERFORMING ORG. REPORT NUMBER P-1318
7. AUTHOR(s) Jeffrey H. Grotte with appendices by James E. Falk and Paul McCoy		8. CONTRACT OR GRANT NUMBER(s) Independent Research
9. PERFORMING ORGANIZATION NAME AND ADDRESS Institute for Defense Analyses Program Analysis Division 400 Army-Navy Drive, Arlington, VA 22202		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE January 1978
		13. NUMBER OF PAGES 102
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

This document is unclassified and suitable for public release.

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

DDC
RECEIVED
AUG 21 1978
B

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Global Optimization, Nonlinear Programming, Separable Programming, Branch and Bound, Minimization, and FORTRAN

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The global optima of nonconvex optimization problems are, in general, impossible to find. Many such problems, however, can be approximated arbitrarily closely by separable problems wherein all functions are piecewise linear. Program MOGG is a FORTRAN code which will find a global optimum to these latter problems. The code is based on a branch and bound algorithm that is guaranteed to terminate after a finite number of steps. The code incorporates a linear programming

→ next page

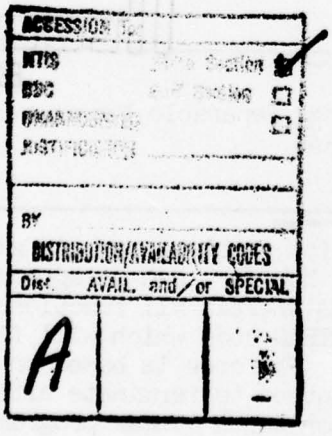
DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. continued

subsystem designed to be numerically stable even for ill-conditioned problems.



(9) Final rept.,

(14)

IDA PAPER P-1318

(6)

A COMPUTER PROGRAM FOR SOLVING SEPARABLE NONCONVEX OPTIMIZATION PROBLEMS

(10) Jeffrey H. Grotte

with appendices by

James E. Falk
Paul F. McCoy

(11) January 1978

(12) 104 p.

(18) IDA/HQ, SRIE

(19) 77-19991, AD-E500 018

IDA

INSTITUTE FOR DEFENSE ANALYSES
PROGRAM ANALYSIS DIVISION
400 Army-Navy Drive, Arlington, Virginia 22202

IDA Independent Research Program

403219

B

FOREWORD

Nonconvex programming computer programs are an essential part of the effective practice of operations research as applied to military, industrial, and economic problems. Many such programs, however, fail to converge, find only local optima, or become unstable when applied to large problems.

This paper documents a computer program that can be applied to a broad range of nonconvex programming problems. The program is important in that it finds a global optimum in a finite number of steps, and has proven to be stable for large problems.

CONTENTS

FOREWORD iii

A. Introduction 1
B. Problems to Which MOGG Applies 3
C. User's Guide 5
D. Sample Problem 9
E. On the Algorithm 13
F. Error Exits 13
G. Variables and Tolerances 13

APPENDICES

A. An Algorithm for Locating Approximate Global Solutions of Nonconvex, Separable Problems -- James E. Falk
B. A Description of the Linear Programming Subroutine LINPRG -- Paul McCoy
C. MOGG Listing

FIGURES

1 A Typical $F_{ij}(x_j)$ 4
2 The Approximating Function $\tilde{F}_{ij}(z_j)$ 4
3 Sample GETPHI 9
4 Data Cards for Sample Problem 10
5 MOGG Sample Output 11
6 MOGG Logic 14

A. INTRODUCTION

Mathematical programming, a fundamental tool of operations research, is frequently used to find solutions to optimization problems arising in the analysis of military, industrial and economic models. The utility of linear programming, applicable to models in which all equations are linear, is well known and one reason for the widespread use of linear programming is the availability of computer codes for solving linear programming problems. Many important problems, however, cannot be conveniently modelled in a linear framework. A brief survey of recent literature reveals nonlinear programming applications to missile allocation, failure diagnosis, media selection for advertising, facility location, chemical process scheduling, design of sewers, and so forth. For these types of analyses, nonlinear optimization problems must be solved.

A major difficulty that arises in nonlinear programming is the existence of local optima. Except when certain convexity conditions obtain, nonlinear programming codes in general cannot guarantee that the answers they produce are globally optimal. Although the use of local optima may be useful in some cases, basing analyses on local optima rather than global optima defeats the purpose of engaging in mathematical programming.

It is therefore noteworthy when a computer code becomes available that can guarantee a global optimum for a large class of nonlinear programming problems--the class of separable, piecewise linear problems--in a finite number of steps. Further, the code can generate piecewise linear approximations to any separable, continuous optimization problem and find a globally optimal solution of the approximate problem. The code has been

tested on a wide range of problems, and the size of problems that can be handled is limited only by computer storage and run time considerations.

The code is based on an algorithm by James E. Falk of The George Washington University. A theoretical treatment of this algorithm is reprinted in Appendix A, which also describes some of the background of this approach. The algorithm uses branch-and-bound to generate a sequence of linear programming sub-problems.

An earlier realization of this algorithm, the NUGLOBAL code[1], was found to have serious stability deficiencies in its linear programming subsection when applied to large problems. The code described in this paper, which is embodied in a program named MOGG, was therefore developed to be stable and also to correct some other, less serious, computational inefficiencies. In particular, a linear programming package designed by John A. Tomlin of Stanford and adapted by Paul F. McCoy of IDA was incorporated into the new code. This linear programming package has proved to be trustworthy.

This paper is divided into seven parts: Part B concisely describes the types of problems the code will solve, and the details of computing piecewise linear approximations to separable, continuous optimization problems; Part C is a user's guide explaining the input necessary to run Program MOGG; a sample problem appears in Part D; Part E presents a flowchart of the algorithm as implemented in this code; Part F remarks on some of the error messages that may be encountered during a MOGG run; and Part G comments on some of the important variables and tolerances used by the code.

---

[1]Hoffman, Karla R, *NUGLOBAL--User's Guide*, Technical Memorandum Serial TM-64866, The George Washington University Program in Logistics, Washington, D.C., March 1975.

Appendix A has already been described. Appendix B is a description of the linear programming package and Appendix C contains a complete FORTRAN listing of Program MOGG.

B. PROBLEMS TO WHICH MOGG APPLIES

Consider the problem P-1

$$\text{P-1}\quad\begin{cases}\text{minimize} & F_1(x)\\ \text{where} & x = (x_1 \cdots x_n)\\ \text{subject to} & F_i(x) \le b_i \qquad i=2,\ldots,q\\ & F_i(x) = b_i \qquad i=q+1,\ldots,m\\ & \ell_j \le x_j \le u_j \qquad j=1,\ldots,n.\end{cases}$$

We will assume that all $F_i(x)$ are continuous over the rectangle $\ell_j \le x_j \le u_j$ $j=1,\ldots,n$ (this condition is actually somewhat stronger than necessary, see Appendix A). With no further restrictions, this problem in general cannot be solved. However, if each $F_i(x)$ is separable, i.e., if each $F_i(x)$ can be written

$$F_i(x) = \sum_{j=1}^{n} F_{ij}(x_j),$$

then we can approximate Problem P-1 by a piecewise linear problem in the following manner. Consider Figure 1 which, we will imagine, depicts some $F_{ij}(x_j)$ for $\ell_j \le x_j \le u_j$. Let us divide the interval $[\ell_j, u_j]$ into t intervals by specifying the points $\{z_j^0, z_j^1, \ldots, z_j^t\}$, which we shall call "cuts" where all that we require is

$$\ell_j = z_j^0 < z_j^1 < \cdots < z_j^t = u_j.$$

Now we define a new function $\tilde{F}_{ij}(z_j)$ for $\ell_j \le z_j \le u_j$ as follows:

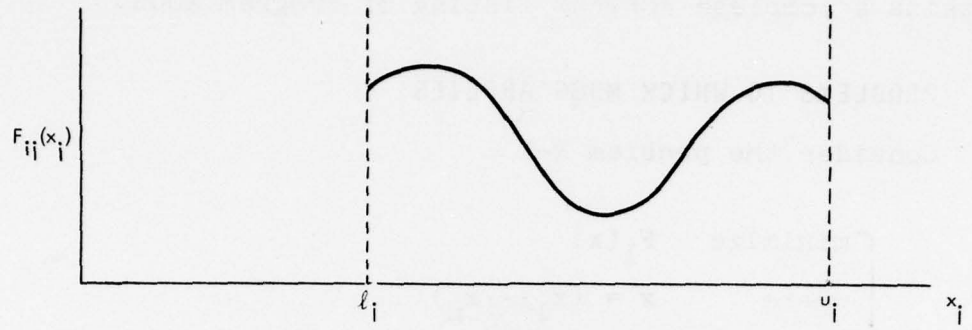


Figure 1. A TYPICAL $F_{ij}(x_j)$

$$\tilde{F}_{ij}(z_j) = \frac{z_j - z_j^k}{z_j^{k+1} - z_j^k} \left(F_{ij}(z_j^{k+1}) - F_{ij}(z_j^k) \right) + F_{ij}(z_j^k)$$

$$\text{for } z_j \in [z_j^k,\ z_j^{k+1}],\quad k=0,\ldots,\ t-1.$$

It is easy to see that $\tilde{F}_{ij}(z_j)$ is continuous and piecewise linear. Figure 2 shows the approximation $\tilde{F}_{ij}(z_j)$ to the function $F_{ij}(x_j)$ of Figure 1, for the choice of $\{z_j^0 \ldots z_j^t\}$ shown (here t=6).

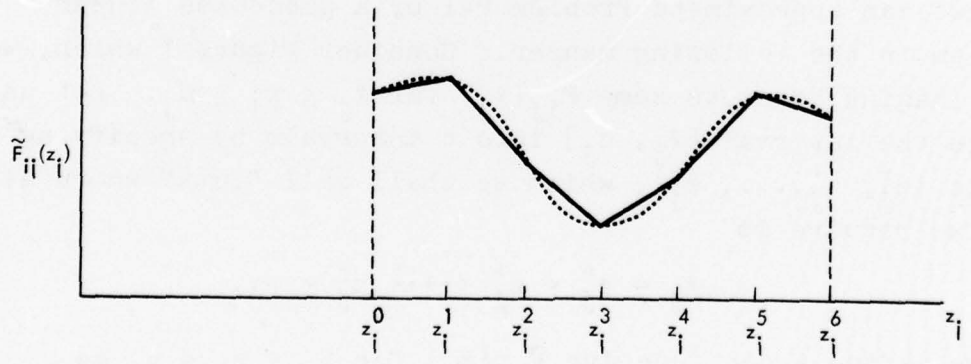


Figure 2. THE APPROXIMATING FUNCTION $\tilde{F}_{ij}(z_j)$

When ℓ_j and u_j are finite, as is often the case in applications, then it follows from the first theorem of Weierstrass that by increasing t, and by judicious choice of the cut points $\{z_j^0 \ldots z_j^t\}$, we can approximate $F_{ij}(x_j)$ arbitrarily closely (according to most of the standard measures of "closeness").

In this way we have constructed an approximation to Problem P-1 which we shall call P-2:

$$\text{P-2} \quad \begin{cases} \text{minimize} & \tilde{F}_1(z) \equiv \sum_{j=1}^{n} \tilde{F}_{1j}(z_j) & \\ \text{where} & z = (z_1 \ldots z_n) & \\ \text{subject to} & \tilde{F}_i(z) \equiv \sum_{j=1}^{n} \tilde{F}_{ij}(z_j) \leq b_i & i=2,\ldots,q \\ & \tilde{F}_i(z) \equiv \sum_{j=1}^{n} \tilde{F}_{ij}(z_j) = b_i & i=q+1,\ldots,n \\ & \ell_j \leq z_j \leq u_j & j=1,\ldots,n. \end{cases}$$

Program MOGG constructs the Problem P-2 from P-1 and finds an optimal solution thereof. Interested readers are referred to Appendix A, which discusses this approach in greater detail and which describes and rigorously justifies the algorithm employed by MOGG.

C. USER'S GUIDE[1]

This section provides the information necessary to use Program MOGG. The notation is from Section A. We make the following conventions. For any variable x_j, if at least one $F_{ij}(x_j)$ is nonlinear, that is, if it is *not* of the form $a_{ij} \cdot x_j$

[1] In this section, Ø will represent "zero" and 0 will represent the letter "oh."

where a_{ij} is a constant, then we will say that x_j is a *nonlinear* variable. Otherwise, we will say that x_j is *linear*. If x_j is linear, then Program MOGG assumes $\ell_j = \emptyset$ and $u_j = +\infty$. When this is not the case, then treating x_j as a nonlinear variable with 1 cut to enforce the upper bound is permissible. Following are the input specifications for MOGG.

Two types of input are required: a user-supplied subroutine and data cards. We describe the subroutine first.

Subroutine GETPHI

One component of input necessary to use MOGG is Subroutine GETPHI (I, J, X, F). Called by MOGG, and given the values of I, J, and X, GETPHI must set F equal $F_{IJ}(X)$. The value of X supplied by MOGG will always equal some cut z_j^k. The value of J will never correspond to a linear variable. At present, no user-supplied read-in capability is provided. It is an elementary matter to modify MOGG to build in such a capability.

Data Cards

Specification Card

Columns	Entry	Format
1-5	NMROWS - the number of rows of Problem P-1, corresponds to m of Section A. Note that this includes the objective function.	I5
6-1Ø	NUMVAR - the number of columns of Problem P-1, corresponds to n.	I5
11-15	MAXLP - the maximum number of calls to the linear programming subsection permitted (1ØØ is a typical choice).	I5
16-2Ø	KBUB, = 1 if an upper bound for the optimal solution is to be provided, otherwise leave blank.	I5

21-25	IXPRIN, = 1 if the user wants printed all feasible points found, otherwise leave blank.	I5
26-3Ø	K1, = 1 if the user wants all LP solutions printed, otherwise leave blank.	I5
31-35	K2, = 1 if the user wishes to see the packed LP matrix at the beginning of the run, otherwise leave blank.	I5
36-4Ø	K3, = 1 will print LP iteration information. Use for debugging only--leave blank for general use.	I5
41-45	K4, = 1 if the user wishes to see the branch and bound list after each stage is completed.	I5
46-5Ø	K5, = 1 if the user would like to scale the LP matrix by dividing each row by a power of 2 near the geometric mean of the largest and smallest (in absolute value) nonzero entries in that row.	I5

Upper Bound Card

This card is included only if KBUB = 1 on the Specification Card. It contains the user-supplied upper bound (Format: F1Ø.6).

Relation Cards

These cards specify the *row type*. Enough cards are necessary to allow 2*NMROWS columns which are considered to be numbered sequentially. Columns 1 and 2 contain bØ (b=blank, Ø=zero). For k=2, ...,NMROWS, columns 2k-1 and 2k contain

-1 if row k of the input problem is an equality,

b1 if row k is an inequality (only $\leq$ is allowed).

Contrary to the notation used for Problem P-1, inequalities and equalities may be listed in any order.

Convexity Cards

These cards contain the *convexity flags*. Enough cards are necessary to provide NMROWS columns, numbered sequentially. Column k contains a Ø if k=1 or if row k of the input problem represents a nonconvex constraint. If row k represents a convex constraint, column k contains a 1. When unsure, the user should use a Ø.

Bound and Cut Cards

For each variable, there is a set of cards as follows. Columns 1-5 of the first card contain the variable number (format I5). These numbers must start at 1 and increase up to NUMVAR. Columns 6-1Ø contain the value of the variable NOINC (format I5). For *linear* variables, NOINC = Ø and no further entries or cards are required. For *nonlinear* variables, NOINC is the number of cuts desired for this variable. NOINC is the same as "t" in Section A. If NOINC ≠ Ø, then columns 11-15 must contain either "AUTO." or "MANU." (format A5). The period must appear. If "AUTO." appears, MOGG will automatically make the cuts. The next card must contain the values of ℓ_j (columns 1-1Ø) and u_j (columns 11-2Ø) for this variable (format 2F1Ø.6). No further cards are then needed. If "MANU." appears, then the values of $z_j^{\emptyset}$ to z_j^{NOINC} must appear, in order, on the next cards. Each z_j^k occupies an F1Ø.6 field. As many cards as necessary are to be used.

Right Hand Side Cards

Enough cards are required to provide NMROWS F1Ø.6 fields. These contain, in order, the right hand sides of Problem P-1 (the b_i). The first field, corresponding to the objective function, must contain Ø.Ø.

Linear Variable Cards

For each linear variable, in order, enough cards are required to provide NMROWS F1Ø.6 fields. The k^{th} field contains the coefficient of the linear variable in row k.

Variable Names Cards

Enough cards are required to provide NUMVAR A5 fields. These contain, in order, alphanumeric names for the variables. If no variable names are desired, then a sufficient number of blank cards must be supplied.

Problem Title Card

Finally, one card must be provided for the problem title. Any alphanumeric expression will do.

D. SAMPLE PROBLEM

This problem is discussed in Section 4 of Appendix A.

$$\text{Minimize} \quad 2x_1^3 - 9x_1^2 + 9x_1 - 2x_2^3 + 9x_2^2 - 9x_2$$

$$\text{subject to} \quad 6x_1^2 - 18x_2 \leq 0$$

$$-6x_1^2 + 18x_2 \leq 9$$

$$0 \leq x_1, x_2 \leq 3.$$

Figure 3 is a listing of the subroutine GETPHI. Figure 4 reproduces the data cards for this problem. Figure 5 shows the MOGG output. This run took 3.7 seconds of CPU time (on a CDC 6400 computer).

```
      SUBROUTINE GETPHI(I,J,X,F)
      F=0.0
      GOTO(100,200,300),I
  100 IF(J.EQ.1)F=2.*X**3-9.*X*X+9.*X
      IF(J.EQ.2)F=(-2.)*X**3+9.*X*X-9.*X
      RETURN
  200 IF(J.EQ.1)F=6.*X*X
      IF(J.EQ.2)F=(-18.)*X
      RETURN
  300 IF(J.EQ.1)F=(-6.)*X*X
      IF(J.EQ.2)F=18.*X
      RETURN
      END
```

Figure 3. SAMPLE GETPHI

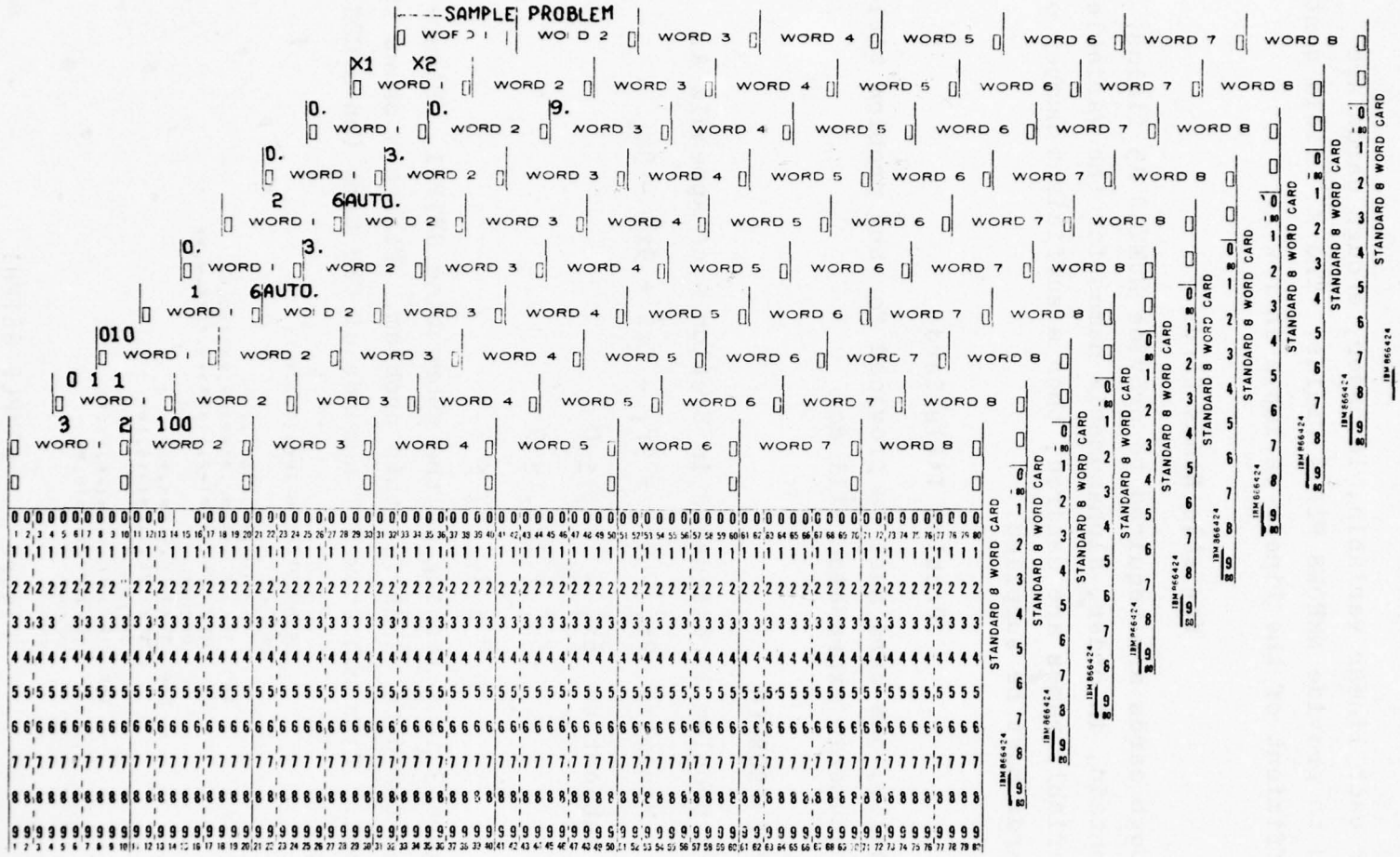


Figure 4. DATA CARDS FOR SAMPLE PROBLEM

```
PROGRAM MOGG--FINDS GLOBAL SOLUTIONS TO APPROXIMATE PROBLEMS

PROBLEM INFORMATION

                          3ROWS
                          2VARIABLES
                        100     LP PROBLEMS WILL BE SOLVED

ROW TYPE--
 0 1 1

CONVEXITY FLAGS--

010

VARIABLE CARDS REPRODUCED--

    1    6AUTO.
0.          3.000
    2    6AUTO.
0.          3.000

RHS CARD(S) REPRODUCED--

  0.          0.          9.000

FOR YOUR INFORMATION          VARIABLE NUMBER          KLO          KRO

                                        1                1            7
                                        2                8           14


STARTING TO ITERATE

    STAGE.PROBLEM        LOWER BOUND          UPPER BOUND       BRANCHING VARIABLE


            0.0              -3.667               -2.667                    2

DONE WITH THIS STAGE
BLB=    -3.667     , BUB=    -2.667     , BRANCHING ON PROBLEM  0.0, VARIABLE NUMBER     2

            1.1            LB GT BUB

            1.2              -3.500               -2.000                    1

            1.3              -3.500               -2.000                    1

DONE WITH THIS STAGE
BLB=    -3.500     , BUB=    -2.667     , BRANCHING ON PROBLEM  1.3, VARIABLE NUMBER     1

            2.1              -2.857               -2.857                    0

            2.2            LB GT BUB

DONE WITH THIS STAGE
BLB=    -3.500     , BUB=    -2.857     , BRANCHING ON PROBLEM  1.2, VARIABLE NUMBER     1

            3.1              -2.857               -2.857                    0

            3.2            LB GT BUB


----SAMPLE PROBLEM

OBJECTIVE FUNCTION AT OPTIMUM    -2.857

VARIABLE VALUES AT OPTIMUM--


   X1              X2
  1.714           1.000
```

Figure 5. MOGG SAMPLE OUTPUT

Note some of the features of the output. The KLO, KRO columns display the limits of the "k-sets" of Appendix A, which are stored as a single variable array. For computational purposes, linear variables are assigned a k-set in which KLO equals KRO. MOGG prints "STARTING TO ITERATE" after completing its data storage routines, and begins the branch and bound procedure. Problems are numbered by their stage and their position in that stage. After completion of each stage, a best lower bound (BLB) and a best upper bound (BUB), if any, are displayed. If no best upper bound is found, BUB will be set equal to 1.E7Ø. If no upper bound is found for an individual problem, the word NONE will appear. In Problem 1.1, LB GT BUB indicates that the lower bound for that branch is greater than the best upper bound presently known, so that no further investigations along that branch will be pursued. Problem 2.1 displays "Ø" as the branching variable to indicate a terminal node of the branch and bound tree.

Additional information can be requested on the specification card. Most of the resulting displays are self explanatory, however, the user should be aware of the following:

- When K1=1, the LP solution will be printed in "packed" form so that basic variables which are equal to zero will be omitted.
- When K2=1, the packed (zeros omitted) matrix will be printed by columns going across the page, with the row number beneath the entry. An identity matrix is annexed to the left of the structural matrix.
- When K3=1, the user should refer to Appendix B for an explanation of the LP iteration printout.
- When K4=1, the column beneath "FLAG" contains the pointer used to divide the k-sets (x^T in Appendix A).

E. ON THE ALGORITHM

Appendix A contains a thorough description of the algorithm. Figure 6 is a flowchart representing the MOGG implementation of this algorithm using some notation from Appendix A. The variable NOLEFT is the number of problems left to solve in any given stage.

The linear programming code used by MOGG is described in Appendix B, and listed in Appendix C. It was chosen for its numerical stability, an important consideration when trying to solve "real world" problems.

F. ERROR EXITS

MOGG makes numerous diagnostic checks throughout its operation and, under some circumstances, will terminate. When this happens, a self-explanatory diagnostic message will be printed along with a reference to the region of the code where the error occurred.

G. VARIABLES AND TOLERANCES

These common blocks provide interroutine communication for MOGG. Block /FIRST/ contains mostly main program variables, while /WORK/ and /BLOCK/ are primarily for the use of the linear programming subsection. Among the important variables are the following (see Section A and Appendix A for terminology):

KLO(I), KRO(I):	These define the lower and upper boundaries of variable I's original k-set.
W:	This array is used by LINPRG to return optimal LP solutions.
LFLG:	LINPRG uses this to indicate infeasibility of a subproblem.
CUTS:	This array stores all the cuts z_j^k.

Figure 6. MOGG LOGIC

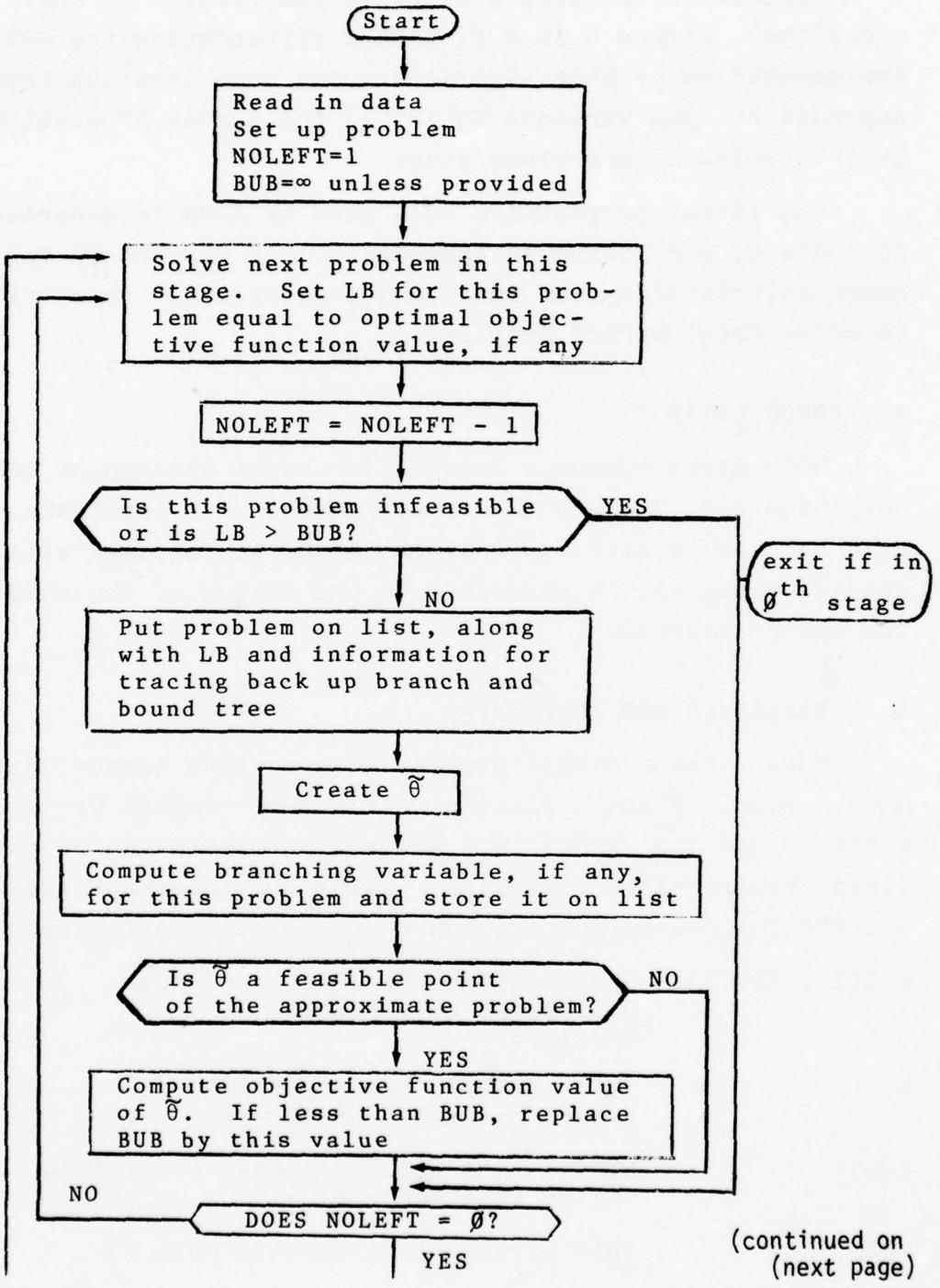
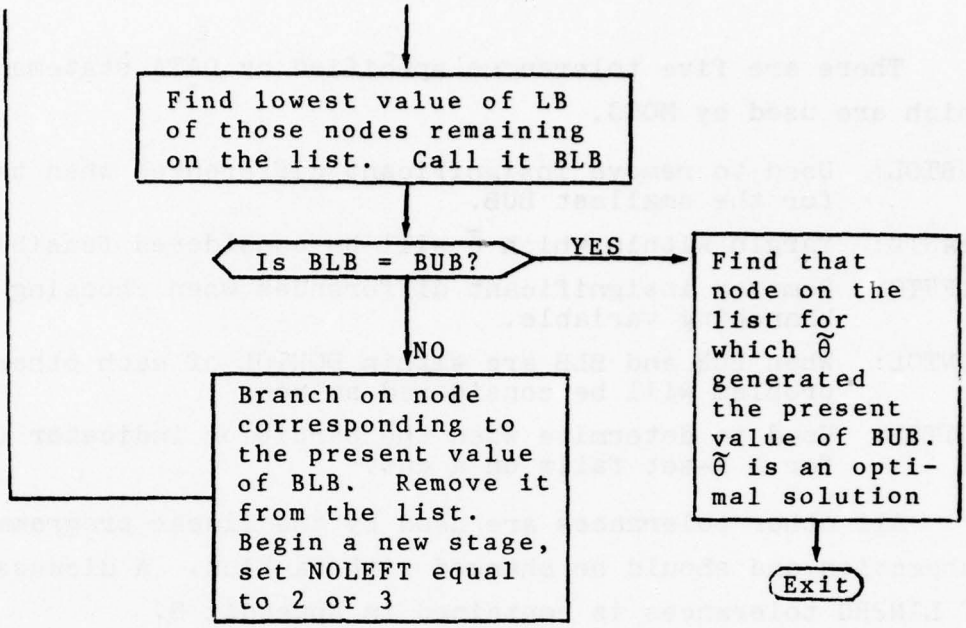


Figure 6. MOGG LOGIC (con't)



ZLSTNO, ZLSTPA, LSTKL, LSTKR, ZLSLB, IBRVR, FLAG: Seven arrays that constitute the list representing the branch and bound tree. ZLSTNO stores stage and problem numbers, ZLSTPA stores the number of the immediate predecessor of each problem, LSTKL and LSTKR are the lower and upper boundaries of the k-sets which distinguish this problem (only the k-sets relating to the predecessor's branching variable are stored). ZLSLB is the objective function value computed for this problem. IBRVR is the branching variable for this problem and FLAG is used to determine the new k-sets when branching on this node.

A, IA: These are used to store the packed LP array.

B: This array stores the right hand side values.

There are five tolerances specified by DATA statements which are used by MOGG.

BUBTOL: Used to remove insignificant differences when testing for the smallest BUB.

FEASTL: Margin within which $\tilde{\theta}$ will be considered feasible.

DIFFTO: Removes insignificant differences when choosing a branching variable.

DONTOL: When BUB and BLB are within DONTOL of each other, the problem will be considered solved.

CUTTOL: Used to determine when the partition indicator (FLAG) for a k-set falls on a cut.

All other tolerances are used by the linear programming subsection and should be changed with caution. A discussion of LINPRG tolerances is contained in Appendix B.

The arrays are presently dimensioned large enough to solve most problems of interest. If the user wishes to redimension the arrays, he is referred to the COMMENT statements at the beginning of the MOGG code (see Appendix C). Note that the variables MAXVAR, MAXCUT, LSTMAX, MAXROW and MAXA must be assigned new values. At present, MOGG can handle

100	Original variables
1100	Total cuts
100	Rows
700	Entries in the branch and bound list
5000	Nonzero elements in the packed linear programming array.

The MOGG routine has performed well on a CDC 6400 with 60-bit words. If round-off problems appear when the code is implemented on machines with smaller words, conversion to double precision is recommended.

APPENDIX A

AN ALGORITHM FOR LOCATING APPROXIMATE GLOBAL SOLUTIONS OF NONCONVEX, SEPARABLE PROBLEMS

James E. Falk

SERIAL T-262

AN ALGORITHM FOR LOCATING APPROXIMATE GLOBAL SOLUTIONS OF NONCONVEX, SEPARABLE PROBLEMS

James E. Falk

April 20, 1972

THE GEORGE WASHINGTON UNIVERSITY
SCHOOL OF ENGINEERING AND APPLIED SCIENCE
INSTITUTE FOR MANAGEMENT SCIENCE AND ENGINEERING
Program in Logistics

1. INTRODUCTION

An algorithm for finding global solutions of nonconvex separable problems was developed by Falk and Soland [3] and Soland [8]. The method is based on the branch and bound philosophy and yields a (generally infinite) sequence of points whose cluster points are global solutions of the problem. The implementation of the method is severely limited by the necessity of computing convex envelopes [4] of the functions involved although a number of applications of the method have been made (e.g., [5], [9]). These applications were possible because of the special structure of the functions involved (e.g., concave or piecewise linear).

The traditional method for treating separable problems involves calculating piecewise linear approximations of the functions defining the problem and applying a modification of the simplex method to the resulting problem (see, e.g., Miller [7]). The modification amounts to a restriction on the usual manner of selecting variables to exchange roles (basic to nonbasic and vice versa) and will yield a local but not necessarily a global solution of the approximating problem.

In this paper we present a method that will yield a global solution of the approximating problem referred to above. The method is similar to the Falk-Soland algorithm but takes advantage of the special structure of the resulting approximate problem and employs the branch and bound philosophy to set up and monitor the solutions of a finite sequence of linear subproblems.

Recently Beale and Tomlin [1] announced that they have developed a similar algorithm which they have incorporated into their UMPIRE mathematical programming system [10]. The basic idea of their method is the same as that of the algorithm detailed herein although their rules for selecting branching nodes and branching variables are different, being developed from an integer programming point of view while ours are modifications of the rules developed in the Falk-Soland method [3] and its extension by Soland [8].

The problem which we address has the form

$$\text{problem Q} \begin{cases} \text{minimize} & F_o(x) \\ \text{subject to} & F_i(x) \leq b_i \qquad i = 1,\ldots,m \\ & \ell \leq x \leq L \end{cases}$$

where ℓ and L are finite lower and upper bounds respectively on x. We assume that each F_i $(i=0,1,2,\ldots,m)$ is separable, i.e.,

$$F_i(x) = \sum_{j=1}^{n} F_{ij}(x_j) \qquad i = 0,1,\ldots,m$$

and that each F_{ij} is continuous. As extension to the case where F_{ij} is piecewise continuous is covered in Section 5.

In Section 2 we define the approximating problem of problem Q and construct the problem obtained by replacing each of the functions involved by their convex envelopes. A related problem is simultaneously introduced and shown to give a sharper underestimate of the optimal value of the approximating problem than does the convex envelope problem. It is this related problem which the branch and bound procedure solves first to get estimates on the optimal value of the approximating problem and to set up new problems if the estimates do not yield a global solution.

A detailed analysis of the complete method is given in Section 3 and an example follows in Section 4. Some computational considerations are given in Section 5.

2. THE APPROXIMATING PROBLEM AND CONVEX ENVELOPES

The approximating problem of the original problem Q is obtained by replacing each function F_{ij} by a piecewise linear approximation over the interval $[\ell_j, L_j]$. One common method (see, e.g., [7]) that is employed involves selecting $p_j + 1$ grid points $y_{jo}, \ldots, y_{jp_j}$ in $[\ell_j, L_j]$ where $y_{jo} = \ell_j$ and $y_{jp_j} = L_j$ and using convex combinations of the numbers $F_{ij}(y_{jk})$ and $F_{ij}(y_{j,k+1})$ as approximations to the values of $F_{ij}(x_j)$ over the subinterval $[y_{jk}, y_{j,k+1}]$. Figure 1 illustrates this type of approximation.

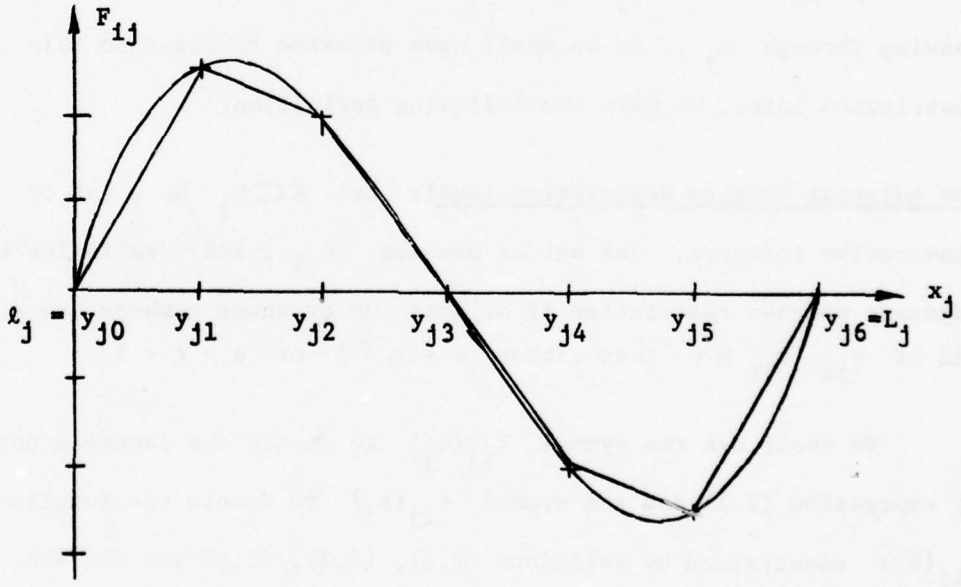


Figure 1. PIECEWISE LINEAR APPROXIMATIONS

Mathematically, we obtain this approximation by setting $F_{ij}(x_j) \tilde{=} f_{ij}(\theta_j)$ where

$$f_{ij}(\theta_j) = \sum_{k\varepsilon K_j} \theta_{jk} F_{ij}(y_{jk}) \tag{2.1}$$

where $K_j = \{0,1,\ldots,p_j\}$, $\theta_j = (\theta_{jo},\ldots,\theta_{jp_j})$ if

$$\sum_{k\varepsilon K_j} \theta_{jk} y_{jk} = x_j \tag{2.2}$$

$$\sum_{k\varepsilon K_j} \theta_{jk} = 1 \tag{2.3}$$

$$\theta_{jk} \geq 0 \qquad k\varepsilon K_j \tag{2.4}$$

and if we add the further restriction that at most two of the weights $\{\theta_{jk} : k\varepsilon K_j\}$ are nonzero, and if two are nonzero, then these must correspond to adjacent grid points. This last restriction is necessary since without it one may obtain any point in the convex hull of the set $\{(y_{jo},F_{ij}(y_{jo})),\ldots(y_{j,p_j},F_{ij}(y_{j,p_j}))\}$ which lies on the vertical line passing through x_j. As we shall have occasion to refer to this restriction later, we make the following definition.

<u>The Adjacent Weights Restriction (AWR)</u>: Let $K \subset K_j$ be a set of consecutive integers. The set of numbers $\{\theta_{jk} : k\varepsilon K\}$ satisfies the adjacent weights restriction if at most two of these numbers are nonzero, and if $\theta_{js}, \theta_{jt} > 0$ then either $s = t - 1$ or $s = t + 1$.

We shall use the symbol $f_{ij}(\theta_j)$ to denote the function defined by expression (2.1) and the symbol $f_{ij}(x_j)$ to denote the function $f_{ij}(\theta_j)$ constrained by relations (2.2), (2.3), (2.4) and the AWR. Thus $f_{ij}(\theta_j)$ denotes a linear function of the variables $\theta_{jo},\theta_{jl},\ldots,\theta_{jp_j}$ while $f_{ij}(x_j)$ denotes a piecewise linear function

of the single variable x_j such as that illustrated in Figure 1.

Likewise

$$f_i(\theta) = \sum_{j=1}^{n} f_{ij}(\theta_j)$$

and

$$f_i(x) = \sum_{j=1}^{n} f_{ij}(x_j)$$

for $i = 0,1,2,\ldots,m$.

By replacing each $F_{ij}(x_j)$ by its piecewise linear approximation $f_{ij}(x_j)$, we obtain the following approximate problem

$$\text{problem P} \begin{cases} \text{minimize} & f_o(\theta) = \sum_{j=1}^{n} \sum_{k\in K_j} \theta_{jk} F_{oj}(y_{jk}) \\ \text{subject to} & f_i(\theta) = \sum_{j=1}^{n} \sum_{k\in K_j} \theta_{jk} F_{ij}(y_{jk}) \leq b_i & (i = 1,\ldots,m) \\ & \sum_{k\in K_j} \theta_{jk} = 1 & (j = 1,\ldots,n) \\ & \theta_{jk} \geq 0 & (j = 1,\ldots,n \; ; \; k\in K_j) \\ & \{\theta_{jk} : k\in K_j\} \text{ satisfies AWR} & (j = 1,\ldots,n) \end{cases}$$

Here $\theta = (\theta_1;\theta_2;\ldots;\theta_n) = (\theta_{10},\ldots,\theta_{1p_1};\theta_{20},\ldots;\ldots;\theta_{n0},\ldots,\theta_{np_n})$.
The solution <u>value</u> of this problem is offered as an approximation to the solution value of the original problem, problem Q. The solution <u>point</u> θ^* of problem P yields an approximation to the solution of problem Q via the relations (2.2), i.e.,

$$x^*_j = \sum_{k\in K_j} \theta^*_{jk} y_{jk} \qquad j = 1,\ldots,n .$$

Problem P is the usual problem that is addressed when seeking solutions of separable programs (see, e.g., [7]). The method of "solution" involves generating a basic feasible solution of the linear

program associated with problem P that satisfies the AWR. A modification of the simplex method is then used to sequentially change the basis until a local solution of problem P is obtained. This modification amounts to a restricted basis entry rule which insures that the AWR are always satisfied by the basic feasible solution associated with each stage of the simplex method. Thus the only nonbasic variables θ_{jk} that may enter the basis at a given iteration are neighbors of existing basic variables. If such a variable is chosen to enter the basis, the outgoing basic variable must be chosen so that the new basic feasible solution satisfies the AWR. It may be shown that this method will yield a local solution of problem P, so that if problem P is convex, the solution will be a global solution. In particular, if problem Q is convex, then so is P and a global solution is assured.

In this paper we are concerned with a method that will produce <u>global</u> solutions of problem P. The method may be considered a specialization of the method of Falk and Soland [3] and the extension described by Soland [8]. In this method it is necessary to compute "convex envelopes" of all functions involved in the problem description over appropriate intervals. A number of convex subproblems are then set up and solved with the branch and bound philosophy monitoring the solution values of these problems and guiding the creation of new subproblems. The <u>convex envelope</u> of a function of a single variable $f_{ij}(x_j)$ over an interval $[\ell_j, L_j]$ is that convex function f_{ij}^c defined over $[\ell_j, L_j]$ such that, if d_{ij} is any convex function on $[\ell_j, L_j]$ which underestimates f_{ij} at every point in $[\ell_j, L_j]$, then d_{ij} also underestimates f_{ij}^c over $[\ell_j, L_j]$. Roughly, the convex envelope of a function is the highest convex function which underestimates that function over the appropriate interval. Alternate and more general definitions and relations concerning convex envelopes are found in [4].

We are interested in determining the convex envelope of the piecewise linear functions $f_{ij}(x_j)$ defined by the relations (2.1) through (2.4) together with the AWR. It is clear geometrically, and not difficult to show analytically, that the convex envelope of this function over $[\ell_j, L_j]$ is the function $f_{ij}^c(x_j)$:

$$f_{ij}^c(x_j) = \min_{\theta_j} \sum_{k\epsilon K_j} \theta_{jk} F_{ij}(y_{jk}) \tag{2.5}$$

$$\text{s.t.} \quad \sum_{k\epsilon K_j} \theta_{jk} y_{jk} = x_j \tag{2.6}$$

$$\sum_{k\epsilon K_j} \theta_{jk} = 1 \tag{2.7}$$

$$\theta_{jk} \geq 0, \; k\epsilon K_j \; . \tag{2.8}$$

Note that we do not impose the AWR on the definition of $f_{ij}^c(x_j)$. We illustrate this definition in Figure 2 which may be compared to Figure 1.

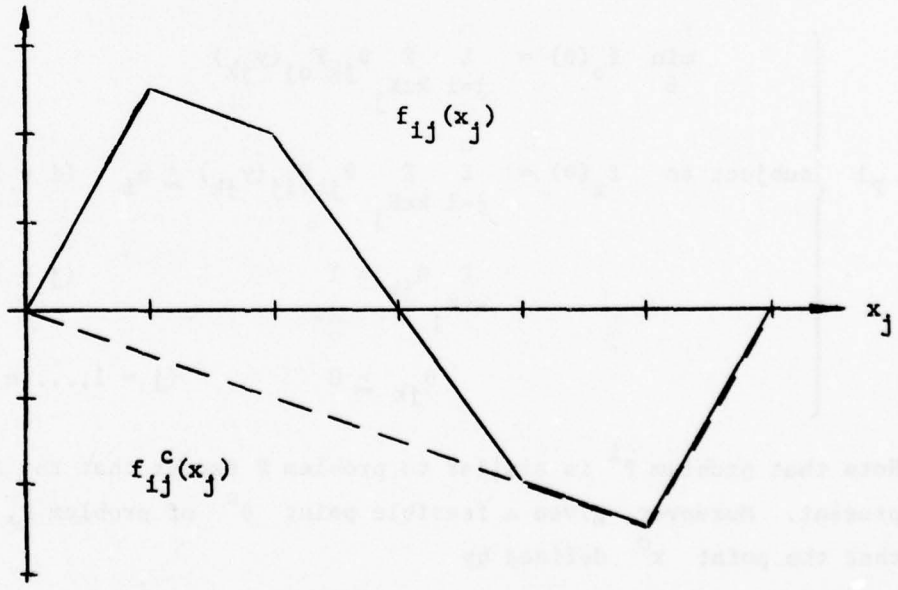


Figure 2. CONVEX ENVELOPES

Thus the calculation of $f_{ij}^c(x_j)$ at a given point x_j involves the solution of a linear program. The first subproblem addressed by the method described in [8] would be

$$\min \quad f_o^c(x) = \sum_{j=1}^{n} f_{oj}^c(x_j)$$

$$\text{subject to} \quad f_i^c(x) = \sum_{j=1}^{n} f_{ij}^c(x_j) \leq b_i \qquad (i = 1,\ldots,m)$$

$$\ell \leq x \leq L \ .$$

This is a convex program whose solution value serves as an underestimate of the solution value of problem P. Because of the piecewise linear nature of the functions f_{ij}^c, it is possible to convert this problem to a linear program. This approach, however, involves explicitly calculating the functions f_{ij}^c for each i and j. Moreover, it would be necessary to do this for a number of problems of the above form. We may avoid these calculations by considering the related linear program:

$$P^1 \quad \begin{cases} \min\limits_{\theta} \quad f_o(\theta) = \sum\limits_{j=1}^{n} \sum\limits_{k\in K_j} \theta_{jk} F_{oj}(y_{jk}) \\ \text{subject to} \quad f_i(\theta) = \sum\limits_{j=1}^{n} \sum\limits_{k\in K_j} \theta_{jk} F_{ij}(y_{jk}) \leq b_i \qquad (i = 1,\ldots,m) \\ \sum\limits_{k\in K_j} \theta_{jk} = 1 \qquad (j = 1,\ldots,n) \\ \theta_{jk} \geq 0 \qquad (j = 1,\ldots,n \ ; \ k\in K_j) \ . \end{cases}$$

Note that problem P^1 is similar to problem P except that the AWR are not present. Moreover, given a feasible point θ^o of problem P, it follows that the point x^o defined by

$$x_j^o = \sum_{k\in K_j} \theta_{jk}^o y_{jk}$$

is feasible for the convex envelope problem by virtue of the inequality

$$f_{ij}^{c}(x_j^o) \leq f_{ij}(\theta^o) .$$

It is, however, possible that the convex envelope problem has feasible points x for which there is no feasible θ satisfying the above expression. For if x is feasible to the convex envelope problem, for each $i = 1,\ldots,m$ there must be a vector i which satisfies conditions (2.6), (2.7), and (2.8) together with the conditions $f_i(\theta) \leq b_i$. This, in itself, does not imply the existence of a single vector which satisfies all of these conditions. On the other hand, any point feasible to problem P is also feasible to P^1 so that the solution value of P^1 offers a valid lower bound on the solution value of P.

3. THE BRANCH AND BOUND ALGORITHM

In this section we present an algorithm to calculate the global solution of problem P which is based on the branch and bound philosophy (see, e.g., [6]). The algorithm considers subsets of a linear polyhedron containing the feasible region $F(P)$ of problem P. A lower bound on the optimal value of problem P is found by minimizing $f_o(\theta)$ over each of these subsets and selecting the smallest of these. A check for solution is made which, if successful, yields a global solution of P. If the check fails, the subset corresponding to the smallest lower bound is further subdivided into either two or three new linear polyhedra and the process continues as before with new and sharper bounds being determined. The process is finite and terminates with a global solution of P.

As is customary with branch and bound procedures, the algorithm is described in terms of a branch and bound tree. (See Figure 6 for an example.) The nodes of the tree will be identified with the symbols N^1, N^2, N^3,... and each node N^i will correspond to a linear subproblem P^i of problem P. It is convenient to also use the notion of a "stage." The first stage of the method consists of problem P^1 (or node N^1) and

its solution. The second stage of the algorithm consists of problems P^1 together with either 2 or 3 new subproblems created from problem P^1. A new stage is created when a previously solved subproblem is chosen for branching and new subproblems are formed. For example, the tree of Figure 6 illustrates that 8 subproblems were formed in 4 stages. The first stage contains node N^1; the second contains nodes N^1, N^2, N^3 and N^4; the third stage contains these nodes and the new nodes N^5 and N^6, and the fourth stage contains nodes N^1 through N^8.

With each node N^t there is associated a linear program of the form

$$\text{problem } P^t \begin{cases} \text{minimize} & f_o(\theta) = \sum_{j=1}^{n} \sum_{k\in K_j} \theta_{jk} F_{oj}(y_{jk}) & \\ \text{subject to} & f_i(\theta) = \sum_{j=1}^{n} \sum_{k\in K_j} \theta_{jk} F_{ij}(y_{jk}) \leq b_i & (i=1,\ldots,m) \\ & \sum_{k\in K_j} \theta_{jk} = 1 & (j=1,\ldots,n) \\ & \theta_{jk} \geq 0 & (j=1,\ldots,n;\ k\in K_j) \\ & \theta_{jk} = 0 & (j=1,\ldots,n;\ k\notin K_j^t) \end{cases}$$

where the sets K_j^t $(j=1,\ldots,n)$ are subsets of consecutive integers of the sets K_j. Note that each problem is a linear program and that these problems differ only in the constraints $\theta_{jk} = 0$ $(j=1,\ldots,n;\ k\notin K_j^t)$. Problem P^1 has $K_j^1 = K_j$ $(j=1,\ldots,n)$ so that problem P^1 resembles problem P except that problem P^1 does not have the AWR imposed on it. Let $F(P^t)$ denote the feasible region of problem P^t and $F(P)$ denote the feasible region of problem P. Note that

$$F(P) \subset F(P^1) \qquad (3.1)$$

and that $F(P^1)$ is a linear polydedron whereas, in general, $F(P)$ is not even a convex set. Assuming $F(P) \neq \emptyset$, problem P^1 will have at least one minimizing point θ^1. In general, let θ^t denote a solution of problem P^t, if one exists, and set

$$LB(t) = \begin{cases} f_o(\theta^t) & \text{if } \theta^t \text{ exists} \\ +\infty & \text{otherwise.} \end{cases}$$

It follows that

$$LB(t) \leq \min \{f_o(\theta) \ : \ \theta \epsilon F(P^t) \cap F(P)\} . \tag{3.2}$$

It is sometimes possible to obtain an upper bound on $f_o(\theta^*)$ from problem P^t. In fact, if $\tilde{\theta}$ is any feasible point to problem P, the number $f_o(\tilde{\theta})$ will be an upper bound on $f_o(\theta^*)$. Using the vector θ^t (assuming it exists) we may, at little computational expense, attempt to construct a vector $\tilde{\theta}^t$ which is feasible to problem P according to the following rule:

Compute the vector x^t using the relationship

$$x_j^t = \sum_{k \epsilon K_j} \theta_{jk}^t y_{jk} \qquad (j=1,\ldots,n) .$$

We then compute a vector $\tilde{\theta}^t$ which satisfies the AWR and the relationship

$$x_j^t = \sum_{k \epsilon K_j} \tilde{\theta}_{jk}^t y_{jk} \qquad (j=1,\ldots,n) .$$

This computation is straightforward since each x_j^t must be in some interval $[y_{j,k'}, y_{j,k'+1}]$ and hence may be expressed as a convex combination of the two adjacent points $y_{j,k'}$ and $y_{j,k'+1}$. If this vector $\tilde{\theta}^t$ also satisfies the constraints $f_i(\theta) \leq b_i$ $(i=1,\ldots,m)$, the number $f_o(\tilde{\theta}^t)$ serves as an upper bound on $f_o(\theta^*)$. We define

the quantity

$$UB(t) = \begin{cases} f_o(\hat{\theta}^t) & \text{if } \hat{\theta}^t \text{ is feasible to P} \\ +\infty & \text{otherwise} \end{cases}$$

so that

$$f_o(\theta^*) \leq UB(t) \tag{3.3}$$

serves as a complementary inequality to (3.2).

In general, the ℓ-th stage of the algorithm consists of problems $P^1,\ldots,P^L$ together with their solutions $\theta^1,\ldots,\theta^L$ (if they exist) and the quantities $LB(1)$, $UB(1),\ldots,LB(L)$, $UB(L)$. A node (or equivalently, a problem) from which no branching has yet taken place (from which no new problems have been created) is termed an <u>intermediate node</u> (<u>intermediate problem</u>). The set of all intermediate problems at stage ℓ is denoted by $I(\ell)$. At stage one, $I(\ell) = \{1\}$, and, if three new problems are created to form stage two, $I(2) = \{2,3,4\}$.

The algorithm is to be constructed in such a way that

$$F(P) \subset \bigcup_{t\in I(\ell)} F(P^t) \ . \tag{3.4}$$

We define the quantities

$$BLB(\ell) = \min_{t\in I(\ell)} \{LB(t)\}$$

and

$$BUB(\ell) = \min_{t=1,\ldots,L} \{UB(t)\} \ .$$

Then (3.2), (3.3) and (3.4) imply that

$$BLB(\ell) \leq f_o(\theta^*) \leq BUB(\ell) \ . \tag{3.5}$$

This is the basic inequality which signals the completion of the algorithm when equality is attained throughout. We will show that our method of branching (creating new problems) sequentially sharpens (3.5) stage by stage and will produce equality in a finite number of stages.

Check for Solution: If $BLB(\ell) = BUB(\ell)$ at the ℓ-th stage, an optimal solution of problem P is $\hat{\theta}^t$ where $UB(t) = f_o(\hat{\theta}^t) = BUB(\ell)$.

If $BLB(\ell) < BUB(\ell)$ we must choose a node N^t for branching, i.e., a problem P^t to create new problems which will sharpen the bounds in (3.5). We shall use the notion that the numbers $LB(t)$ represent approximations to the quantities $\min \{f_o(\theta) : \theta \epsilon F(P^t) \cap F(P)\}$. Since we are interested in determining $\min \{f_o(\theta) : \theta \epsilon F(P)\}$, we choose the smallest of the numbers $LB(t)$ to determine P^t, the problem most likely to generate a global solution of P.

Choice of Branching Node: Choose an intermediate node N^T for further branching where $LB(T) = BLB(\ell)$.

Actually the algorithm will converge if any intermediate node is selected for further branching and it is sometimes convenient from a computational point of view to use a different rule for branching. A common alternative is to select that problem which has been solved last for further analysis, since the data defining that problem are on hand and data needed for the new problem are very similar. This alleviates the bookkeeping involved and tends to minimize the number of times a particular branch in the tree is revisited. On the other hand, the tree tends to grow larger than the tree our rule would grow and would not be efficient if the total time required is largely a function of the time required to solve the subproblems. In our application, the amount of data required to distinguish one problem from another is minimal so that this should not be a factor.

Having selected node N^T for branching at stage ℓ , we create new subproblems by choosing a branching variable θ_J (or, equivalently, x_J) and partitioning the set K_J^T into subsets of consecutive integers. The rule for selecting x_J follows.

Choice of a Branching Variable: Compute each of the differences

$$\sum_{k\in K_j} (\tilde{\theta}_{jk}^T - \theta_{jk}^T) F_{ij}(y_{jk}) \qquad (3.6)$$

for $i = 0,1,\ldots,m$ and $j = 1,\ldots,n$. Select J which corresponds to the largest of these differences.

If all of these differences were nonpositive, upon summing over j for each $i = 0,1,\ldots,m$ we obtain

$$f_i(\tilde{\theta}^T) - f_i(\theta^T) \leq 0 \qquad (i=0,\ldots,m) .$$

Thus

$$f_o(\tilde{\theta}^T) \leq f_o(\theta^T) = BLB(\ell)$$

and

$$f_i(\tilde{\theta}^T) \leq f_i(\theta^T) \leq b_i \qquad (i=1,\ldots,m)$$

Since $\tilde{\theta}^T$ satisfies the AWR, we see that $\tilde{\theta}^T \in F(P)$ so that

$$BUB(\ell) \leq f_o(\tilde{\theta}^T) \leq BLB(\ell)$$

that is, $\tilde{\theta}^T$ must have been a global solution of problem P, contradicting our previous assumption. Thus, unless we are at a solution, at least one of the differences (3.6) is positive and we choose J corresponding to the largest of these quantities.

This rule for selecting a branching variable is analogous to the rule suggested in [3] and [8]. Since, at a solution, all differences (3.6) will be nonpositive, we are selecting a variable corresponding to the worst violation of this criterion.

Note that not all differences (3.6) need be calculated at every stage since some will automatically be zero. If the set $\{\theta_{jk} : j\in K_j^T\}$ satisfies the AWR, for some j then $\theta_j^T = \tilde{\theta}_j^T$ so that all of the corresponding differences (3.6) for $i = 0,\ldots,m$ are zero. Moreover,

if $F_{ij}(x_j)$ is a convex function, the piecewise linear approximation $f_{ij}(x_j)$ of it (equations (2.1) through (2.4) and the AWR) will also be convex. If we denote this approximation by $\tilde{F}_{ij}(x_j)$ we have

$$\sum_{k\in K_j} \tilde{\theta}^T_{jk} F_{ij}(y_{jk}) = \sum_{k\in K_j} \tilde{\theta}^T_{jk} \tilde{F}_{ij}(y_{jk})$$

$$= \tilde{F}_{ij}\Big(\sum_{k\in K_j} \tilde{\theta}^T_{jk} y_{jk}\Big) \qquad (\tilde{\theta}^T_j \text{ satisfies AWR})$$

$$= \tilde{F}_{ij}\Big(\sum_{k\in K_j} \theta^T_{jk} y_{jk}\Big)$$

$$\leq \sum_{k\in K_j} \theta^T_{jk} F_{ij}(y_{jk})$$

so that the corresponding differences (3.6) are automatically nonpositive. Incidentally, this also proves that the algorithm yields a global solution of a convex program in a single stage.

Having selected variable J for branching, we now are in a position to create the new problems of the $(\ell+1)$-st stage. Let $K^T_J = \{p,p+1,\ldots,q,\ldots,r\}$. Note $x^T_J \neq y_{Jp}$ since in this case $\theta^T_{Jp} = 1$ while $\theta^T_{J,p+1} = \ldots \approx \theta^T_{Jr} = 0$ and the difference (3.6) would be zero. Likewise $x^T_J \neq y_{Jr}$. Note also that K^T_J contains at least three indices for otherwise branching could not take place on this variable. We may assume that $x^T_J \in [y_{Ja}, y_{Jb}]$ where y_{Ja} is the nearest left neighboring division point of x^T_J and y_{Jb} is the nearest right division point. We do not exclude the case where $x^T_J = y_{Ja} = y_{Jb}$, i.e., where x^T_J falls on a division point.

Recall that problems $P^1,\ldots,P^L$ have been set up and solved at the end of stage ℓ.

Branching Rule (refer to Figure 3):

$$\text{Let } K_J^- = \{k : k\varepsilon K_J^T \text{ and } y_{Jk} \leq y_{Ja}\}$$

$$K_J^o = \{a,b\}$$

$$K_J^+ = \{k : k\varepsilon K_J^T \text{ and } y_{Jb} \leq y_{Jk}\}$$

Referring to the general definitions of problem P^t at the beginning of this section, define a new problem P^t by setting K_J^t equal to one of the above sets if that set contains at least two elements. The other index sets K_j^t are unchanged (i.e., $K_j^T = K_j^T$ $(j\neq J)$). In this manner we may define at least two new problems (since K_J^T had at least three points and $y_{Jp} \neq x_J^T$, $y_{J-} \neq x_J^T$) and possibly three new problems. These problems are numbered P^{L+1}, P^{L+2} and P^{L+3} (if defined). Note that if $a \neq b$, the problem whose index set $K_J^t = \{a,b\}$ must have only solutions with θ_J^t satisfying the AWR. The various possibilities are illustrated by example in Figure 3. Only the first possibility yields three new problems.

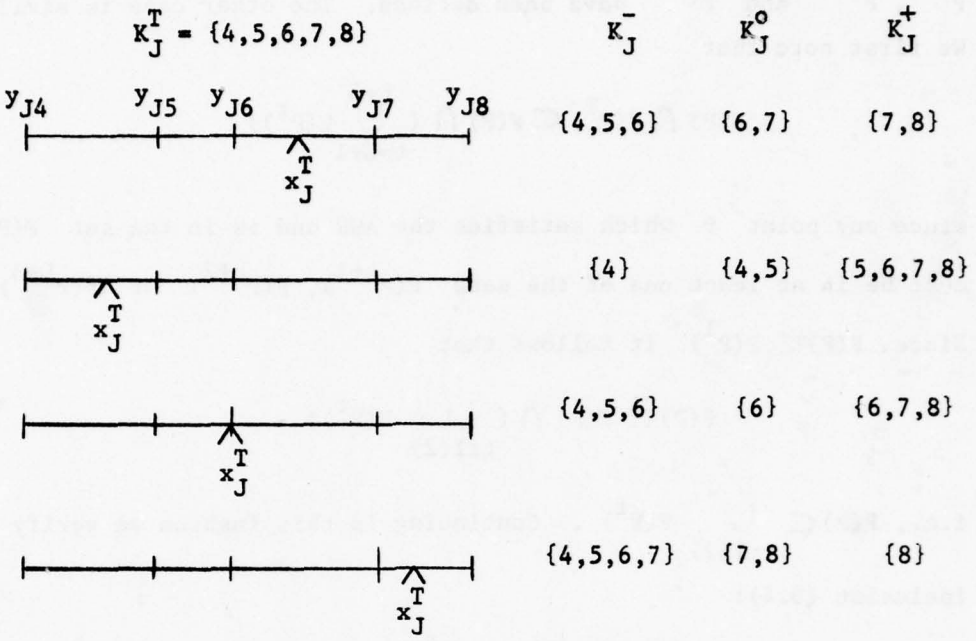


Figure 3. BRANCHING ON VARIABLE x_J

Beale and Tomlin suggest a different branching rule wherein two new subproblems are defined at each stage. Using the above notation, they set

$$K_J^- = \{k : k \epsilon K_J^T \text{ and } y_{Jk} \leq y_{Jb}\}$$

$$K_J^+ = \{k : k \epsilon K_J^T \text{ and } y_{Ja} \leq y_{Jk}\}$$

so that the feasible regions of their problems P^{k+1} and P^{k+2} overlap somewhat more than ours do. Referring to Figure 3, their sets K_J^- and K_J^+ would be $\{4,5,6,7\}$ and $\{6,7,8\}$ in the first case while in the other three cases, their sets would define the same subproblems as we do.

In the remarks which follow we shall assume that these problems P^{L+1}, P^{L+2} and P^{L+3} have been defined. The other case is similar. We first note that

$$F(P) \cap F(P^T) \subset F(P) \cap \left(\bigcup_{t=L+1}^{L+3} F(P^t) \right)$$

since any point θ which satisfies the AWR and is in the set $F(P^T)$ must be in at least one of the sets $F(P^{L+1})$, $F(P^{L+2})$ or $F(P^{L+3})$. Since $F(P) \subset F(P^1)$ it follows that

$$F(P) \subset F(P) \cap \left(\bigcup_{t \in I(2)} F(P^t) \right)$$

i.e., $F(P) \subset \bigcup_{t \in I(2)} F(P^t)$. Continuing in this fashion we verify inclusion (3.4):

$$F(P) \subset \bigcup_{t \in I(\ell)} F(P^t) \tag{3.4}$$

Moreover, since any point in one of the sets $F(P^{L+1})$, $F(P^{L+2})$ or $F(P^{L+3})$ must lie in $F(P^T)$ we have

$$\bigcup_{t \in I(\ell)} F(P^t) \subset \bigcup_{t \in I(\ell-1)} F(P^t) .$$

This inclusion must be strict since the point θ^T cannot lie in any of the sets $F(P^{L+1})$, $F(P^{L+2})$ or $F(P^{L+3})$. For suppose $\theta^T \in F(P^{L+1})$ and $K_J^{L+1} = \{p, \ldots, a\}$. Then $\theta^T_{Jk} = 0$ for $k = a+1, \ldots, r$ and $x_J^T < y_{Ja}$ which contradicts the assumption that $x_J^T \in [y_{Ja}, y_{Jb}]$.

These remarks yield

$$F(P) \subset \bigcup_{t \in I(\ell)} F(P^t) \subsetneq \bigcup_{t \in I(\ell-1)} F(P^t) \subsetneq \ldots \subsetneq F(P^1) \tag{3.7}$$

i.e., the sets $\bigcup_{t\in I(\ell)} F(P^t)$ are converging monotonically towards the set $F(P)$.

When new problems are created for the $(\ell+1)$-st stage, new lower and upper bounds are calculated. Note that

$$\min \{LB(P^{L+1}), LB(P^{L+2}), LB(P^{L+3})\} \geq LB(P^T)$$

since $F(P^T) \not\supseteq \bigcup_{t=L+1}^{L+3} F(P^t)$. Moreover, since the point θ^T for which $f_o(\theta^T) = LB(P^T)$ is not feasible for the new problems, it is likely that the above inequality is strict. The above inequality, together with the definitions of $BLB(\ell)$ and $BUB(\ell)$ yield

$$BLB(1) \leq \ldots \leq BLB(\ell) \leq f_o(\theta^*) \leq BUB(\ell) \leq \ldots \leq BUB(1) \qquad (3.8)$$

so that the upper and lower bounds are converging towards the optimal value of P . It remains to show that the process converges in a finite number of stages.

Theorem. After a finite number of stages, the algorithm yields a global solution of problem P.

Proof. At each stage of the algorithm an index $J\in\{1,\ldots,N\}$ is selected and the set K_J^T is subdivided into either two or three new sets of consecutive integers according to the branching rule. Each of these new sets contains at least two integers. Since there are but a finite number of choices for J and a finite number of ways of subdividing the original sets K_j into sets containing at least two consecutive integers, the algorithm would (if it continued) eventually produce problems whose feasible regions contained only points which satisfy AWR (i.e., eventually $F(P) = \bigcup_{t\in I(\ell)} F(P^t)$) . Such problems must be intermediate problems since their regions cannot be further decomposed, and $LB(t) = UB(t)$. Thus equality must eventually occur in (3.8) and the algorithm is finite.

4. AN EXAMPLE

Problem Q:

$$\text{minimize} \quad F_o(x) = (2x_1^3 - 9x_1^2 + 9x_1) + (-2x_2^3 + 9x_2^2 - 9x_2)$$

$$\text{subject to} \quad F_1(x) = -6x_1^2 + 18x_2 \leq 9$$

$$F_2(x) = 6x_1^2 - 18x_2 \leq 0$$

$$0 \leq x_1, x_2 \leq 3$$

The feasible region of this problem is sketched in Figure 4. There are local solutions near the points (0,0.5), (1.787,1.065) and (2.738,3.000) with values -2.50, -2.97 and -1.46 respectively. The subdivision points are taken at intervals of 1/2 starting at 0. These values and the values of functions F_{ij} at these points are displayed in Table 1 and the results of linear approximations are sketched in Figure 5.

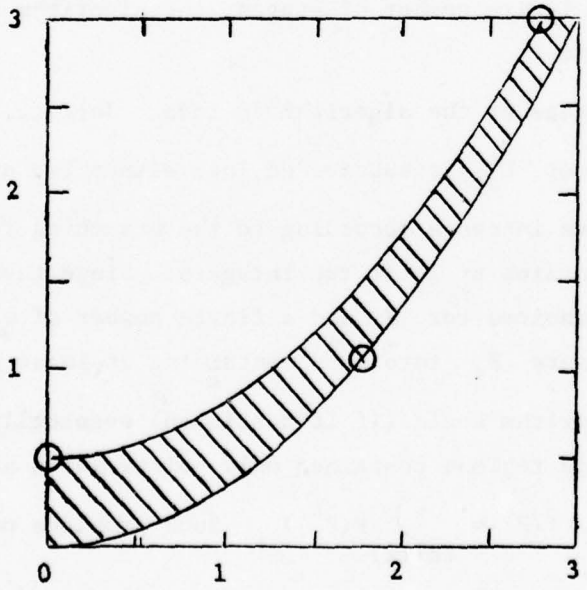
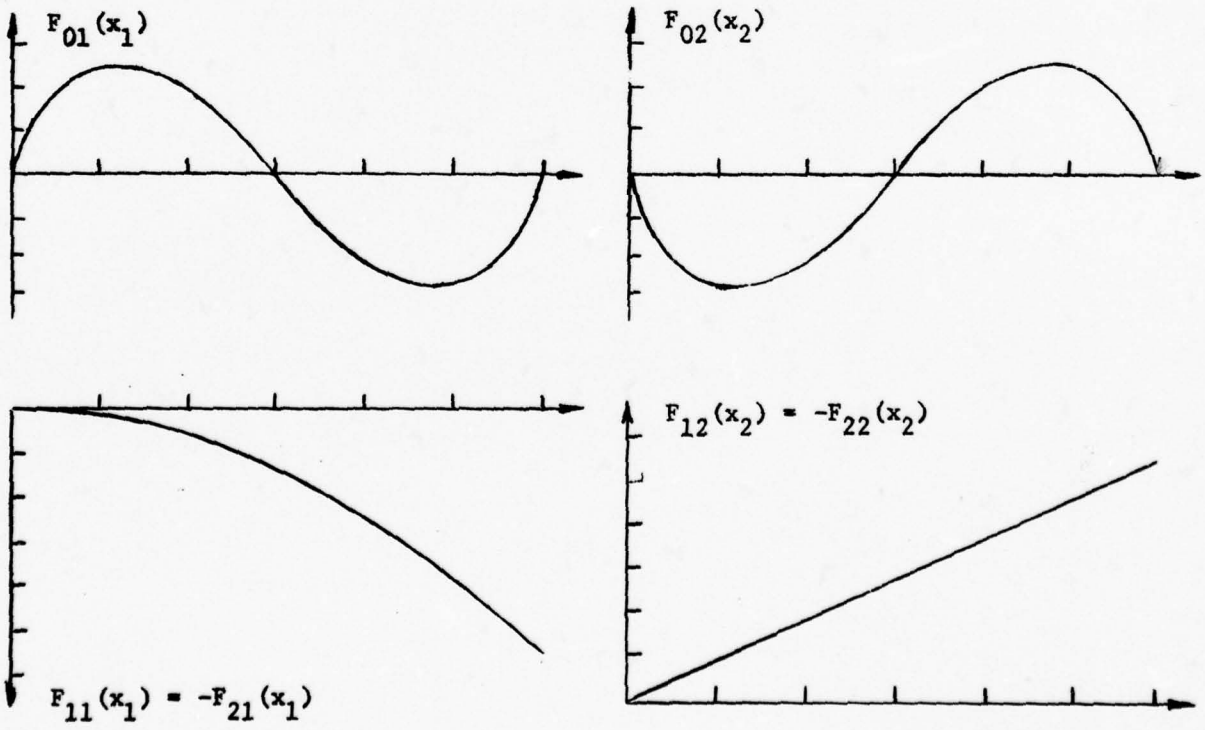


Figure 4. FEASIBLE REGION FOR EXAMPLE

Table 1.

DATA FOR EXAMPLE

x_{1k}	F_{01}	F_{11}	F_{21}	x_{2k}	F_{02}	F_{12}	F_{22}
0	0	0	0	0	0	0	0
1/2	5/2	-3/2	3/2	1/2	-5/2	9	-9
1	2	-6	6	1	-2	18	-18
3/2	0	-27/2	27/2	3/2	0	27	-27
2	-2	-24	24	2	2	36	-36
5/2	-5/2	-75/2	75/2	5/2	5/2	45	-45
3	0	-54	54	3	0	54	-54



Each subproblem has variable $\theta = (\theta_1,\theta_2) = (\theta_{10},\ldots,\theta_{16};$ $\theta_{20},\ldots,\theta_{26})$. The data provided by subproblems is given in Table 2 and the branch and bound tree is illustrated in Figure 6. The global solution of the approximate problem is found to be the point

$$x^* = (1.714,1.000)$$

with objective function value -2.857. This solution is actually found at node 6 but not recognized until problem 8 has been solved.

Table 2.
SOLUTION VALUES FOR EXAMPLE

stage	problem	index sets	solution	solution value	vector x	vector $\tilde{\theta}$	objective function	branching variable	best bounds
ℓ	p^t	K_1^t K_2^t	θ_1^t θ_2^t	$f_o(\theta^t)=$ LB(t)	x_1^t x_2^t	$\tilde{\theta}_1^t$ $\tilde{\theta}_2^t$	$f_o(\tilde{\theta}^t)=$ UB(t)	x_J	BLB(k) BUB(k)
1	1	{0,1,2,3,4,5,6}	0 0 0 0 1 0 0	-3.667	2.000	0 0 0 0 1 0 0	-2.667	x_2	-3.667
		{0,1,2,3,4,5,6}	0 0 5/6 0 0 0 1/6		1.333	0 0 1/3 2/3 0 0 0			-2.667
2	2	{0,1,2,3,4,5,6}	1/4 0 0 0 3/4 0 0	-3.500	1.500	0 0 0 1 0 0 0	-2.000	x_1	
		{0,1,2}	0 0 1 * * * *		1.000	0 0 1 0 0 0 0			
	3	{0,1,2,3,4,5,6}	1/4 0 0 0 3/4 0 0	-3.500	1.500	0 0 0 1 0 0 0	-2.000	x_1	
		{2,3}	* * 1 0 * * *		1.000	0 0 1 0 0 0 0			
	4	{0,1,2,3,4,5,6}	0 0 0 0 0 1 0	-2.500	2.500	0 0 0 0 0 1 0	-0.416	x_2	-3.500
		{3,4,5,6}	* * * 11/18 0 0 7/18		2.083	0 0 0 0 5/6 1/6 0			-2.667
3	5	{0,1,2,3}	1 0 0 0 * * *	-2.500	0.000	$\tilde{\theta}^5 = \theta^5$	-2.500	—	
		{0,1,2}	0 1 0 * * * *		0.500				
	6	{3,4,5,6}	* * * 4/7 3/7 0 0	-2.857	1.714	$\tilde{\theta}^6 = \theta^6$	-2.857	—	-3.500
		{0,1,2}	0 0 1 * * * *		1.000				-2.857
4	7	{0,1,2,3}	0 0 0 1 * * *	-2.000	1.500	$\tilde{\theta}^7 = \theta^7$	-2.000	—	
		{2,3}	* * 1 0 * * *		1.000				
	8	{3,4,5,6}	* * * 4/7 3/7 0 0	-2.857	1.714	$\tilde{\theta}^8 = \theta^8$	-2.857	—	-2.857
		{2,3}	* * 1 0 * * *		1.000				-2.857

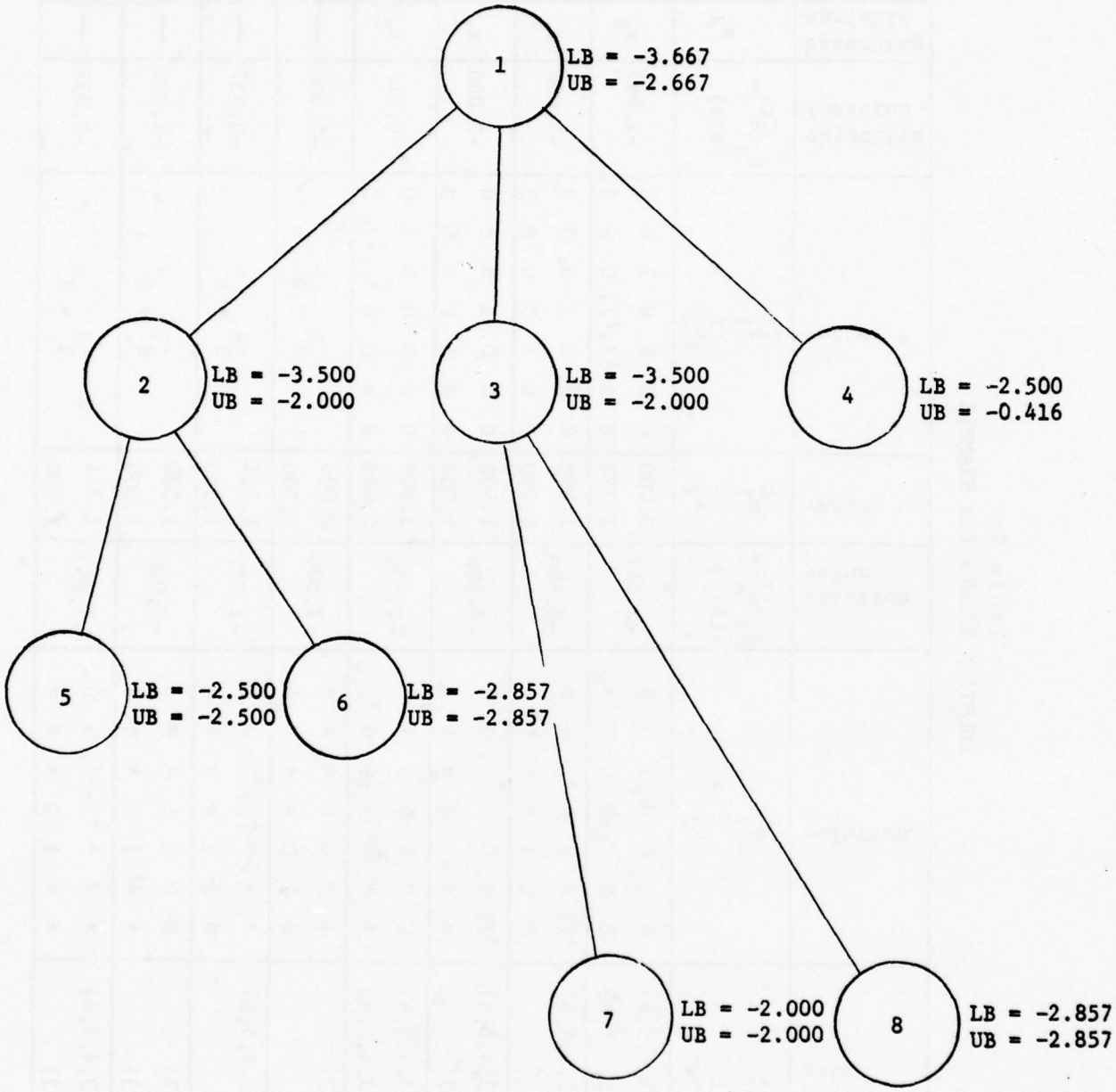


Figure 6. BRANCH AND BOUND TREE FOR EXAMPLE

5. SOME COMPUTATIONAL CONSIDERATIONS AND EXTENSIONS

In this section we point out some computational aspects of the method, some possible variations, and an extension to non-continuous problems.

We first note that each problem P^t contains m constraints corresponding to the m constraints of problem Q plus n constraints of the form $\Sigma\, \theta_{jk} = 1$. Thus the Generalized Upper Bounding Technique of Dantzig and Van Slyke [2] may be used to advantage here, and especially if n is large compared to m. This method allows one to maintain a basis of size $m \times m$.

Since each problem P^t is distinguished by the sets K_j^t, one need carry in memory only that information which identifies these sets, e.g., the first and last indices of the sets. Beale and Tomlin [1] refer to these indices as "flags". The matrix identifying the coefficients of the objective function and the first m constraints of P is common to all problems P^t. Since the basic solution of a problem being branched from is not feasible to the newly created problems, it is not clear that the basis of each problem P^t should be carried in memory along with the sets K_j^t. On the other hand, the basic solution of a problem being branched from only fails to be feasible to its descendants by virtue of one constraint and hence may be useful in creating basic feasible solutions to the newly created problems.

Once a point $\theta^{(q)}$ is found which is feasible to problem P^t, one could attempt to produce a feasible solution $\tilde{\theta}^{(q)}$ which satisfies the AWR by the device outlined in Section 3. The computations necessary to produce such a point are fairly simple. If such a point $\tilde{\theta}^{(q)}$ may be produced, one can immediately compute $f_o(\tilde{\theta}^{(q)})$ and compare this

with the $BUB(\ell)$, updating this number if $f_o(\hat{\theta}^{(q)}) < BUB(\ell)$. In such a way one may be able to tighten the number $BUB(\ell)$ at each simplex iteration solving P^t and possibly come across an optimal solution θ^* of P during the solution of a subproblem P^t. Of course, this solution would not be recognized as such until equality occurs in (3.8).

Finally, we point out a simple modifcation of the method that will allow one to deal with piecewise continuous functions F_{ij}. In order to insure that problem Q has a solution, we assume also that each F_{ij} is lower semicontinuous. The grid points $\{y_{jk} : k \in K_j; j=1,\ldots,n\}$ are chosen so that all points of discontinuity of the F_{ij}'s are among them. Let y_{JK} be a point of discontinuity of F_{LJ} and set

$$F_{IJ}^- = \lim_{x_J \uparrow y_{JK}} F_{IJ}(XJ)$$

$$F_{IJ}^o = F_{IJ}(y_{JK})$$

$$F_{IJ}^+ = \lim_{x_J \downarrow y_{JK}} F_{IJ}(XJ)$$

The lower semicontinuity of F_{IJ} at y_{JK} implies that $F_{IJ}^o \leq \min \{F_{IJ}^-, F_{IJ}^+\}$. Assume, for the sake of discussion, that strict inequality holds, and define new indices K^-, K^o and K^+ corresponding to the quantities F_{IJ}^-, F_{IJ}^o and F_{IJ}^+ respectively. These indices are to be ordered as

$$K - 1 < K^- < K^o < K^+ < K + 1$$

and corresponding new variables θ_{JK}^-, θ_{JK}^o and θ_{JK}^+ are defined.

Problem P is thus redefined with $\theta^-_{JK} F^-_{iJ} + \theta^o_{JK} F^o_{iJ} + \theta^+_{JK} F^+_{iJ}$ replacing $\theta_{JK} F_{iJ}(y_{JK})$, $\theta^-_{JK} + \theta^o_{JK} + \theta^+_{JK}$ replacing θ_{JK} and $\{\ldots, K-1, K^-, K^o, K^+, K+1, \ldots\}$ replacing K_J.

With these modifications carried out at every point of discontinuity, the algorithm may be applied as before with no additional changes. Note that a global solution of problem P cannot have adjacent nonzero pairs $(\theta^-_{JK}, \theta^o_{JK})$ or $(\theta^o_{JK}, \theta^+_{JK})$ unless the value of $F_{oJ}(y_{JK})$ is zero, for otherwise the value of f_o could be decreased by setting $\theta^o_{JK} = 1$ while still maintaining feasibility. Even if $F_{oJ}(y_{JK}) = 0$ and one of the above pairs is nonzero, an equivalent feasible solution may be found for which $\theta^o_{JK} = 1$ and which gives the same value to $f_o(\theta)$.

In the case that $F^o_{IJ} = F^-_{IJ}$ (F_{IJ} is continuous from the left), one need only define two new variables, say θ^o_{JK} and θ^+_{JK}, and modify problem P as above. The case where F_{IJ} is right continuous is similar.

REFERENCES

[1] BEALE, E. M. L. and TOMLIN, J. A. (1970). Special facilities in a general mathematical programming system for nonconvex problems using ordered sets of variables. Proceedings of the Fifth International Conference on Operations Research (J. Lawrence, ed.) 447-454. Tavistock Publications, London.

[2] DANTZIG, G. B. and VAN SLYKE, R. M. (1967). Generalized upper bounding techniques. J. Comput. System Sci. 1 213-226.

[3] FALK, J. E. and SOLAND, R. M. (1969). An algorithm for separable nonconvex programming problems. Management Sci. 15 550-569.

[4] FALK, J. E. (1969). Lagrange multipliers and nonconvex programs. SIAM J. Control 7 534-545.

[5] FALK, J. E. and HOROWITZ, J. L. (1972). Critical path problems with concave cost-time curves. Paper submitted for publication.

[6] LAWLER, E. L. and WOOD, D. E. (1966). Branch-and-bound methods: A survey. Operations Res. 14 699-719.

[7] MILLER, C. E. (1963) The simplex method for local separable programming. Recent Advances in Mathematical Programming (R. L. Graves and P. Wolfe, eds.) 89-100. McGraw Hill, New York.

[8] SOLAND, R. M. (1971). An algorithm for separable nonconvex programming problems II: Nonconvex constraints. Management Sci. 17 759-773.

[9] SOLAND, R. M. (1971). Optimal plant location with concave costs. Paper presented at 39th National Meeting of the Operations Research Society of America in Dallas, Texas.

[10] TOMLIN, J. A. (1970). Branch and bound methods for integer and non-convex programming. Integer and Nonlinear Programming (J. Abadie, ed.) 437-450. North Holland Publishing Company, Amsterdam.

APPENDIX B

A DESCRIPTION OF THE LINEAR PROGRAMMING SUBROUTINE LINPRG

Paul F. McCoy

A. INTRODUCTION

The subroutine LINPRG solves linear programming problems by the standard product form version of the simplex method, as described in [1]. LINPRG is a slight modification of the code written by John Tomlin to run the experiments presented in [4]. It was used again for the tests in [2]. An important feature of the code is that basis reinversion is accomplished by LU decomposition using Gaussian elimination. The reinversion algorithm was developed by Tomlin and is described in [6]. It uses a pivot tolerance in choosing the pivot elements so as to compromise the goals of minimizing the creation of non-zero elements and of pivoting on large elements to maintain numerical stability.

B. INTERNAL WORKINGS OF LINPRG

Reference [3] provides background reference for this section.

NOTATIONS

NCOL = number of variables (including structurals, slacks and artificials),

NROW = number of rows (including the objective row),

x = the (NCOL - NROW) vector of structural variables,

s = the NROW vector of slack and artificial variables,

c = the (NCOL - NROW) vector of costs (objective function coefficients),

A = the [(NROW - 1) x (NCOL - NROW)] matrix of structural coefficients,

b = the (NROW - 1) vector of right hand side values corresponding to the linear constraints.

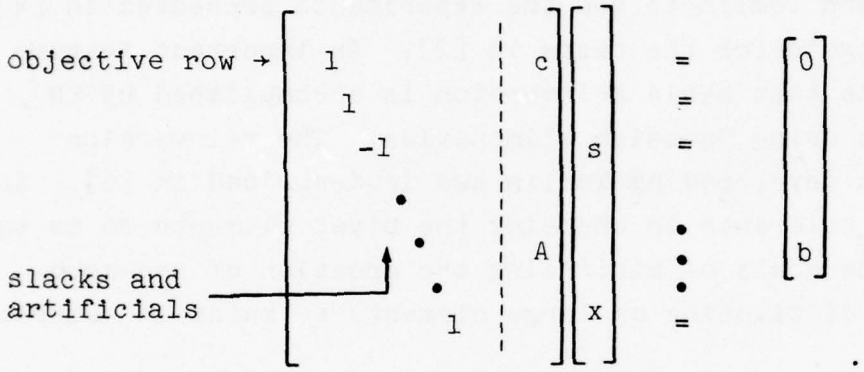
BASIC PROBLEM

minimize cx such that

$$Ax \begin{pmatrix} = \\ \leq \\ \geq \end{pmatrix} b \qquad \text{and } x \geq 0 .$$

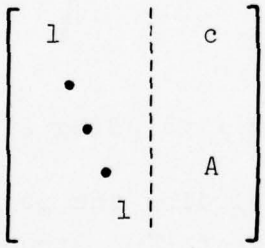
ACTUAL PROBLEM

maximize s_1 such that $s \geq 0$, $x \geq 0$ and



SET-UP PROCEDURES

Before calling LINPRG, MOGG packs the constraint coefficients



into the one-dimensional array A(·). Only non-zero entries are stored. The location of coefficients is maintained by the row index array IA(·) and the column pointer array LA(·).

A(NELEM) = value of NELEMth nonzero coefficient,

IA(NELEM) = row of that coefficient,

LA(NCOL) = the first element of A(·) belonging to column NCOL,

LA(NCOL + 1) - 1 = the last element belonging to column NCOL.

The objective coefficients are placed in the first row. The right hand side coefficients $\begin{bmatrix} 0 \\ b \end{bmatrix}$ are stored in unpacked form in the array B(·). The type of each row is stored in array ISTYPE(·):

$$\text{ISTYPE(ROW)} = \begin{cases} 0 \text{ if ROW} = 1 \text{ (objective row)} \\ -1 \text{ if equality } (=) \\ 1 \text{ if inequality } (\leq \text{ or } \geq). \end{cases}$$

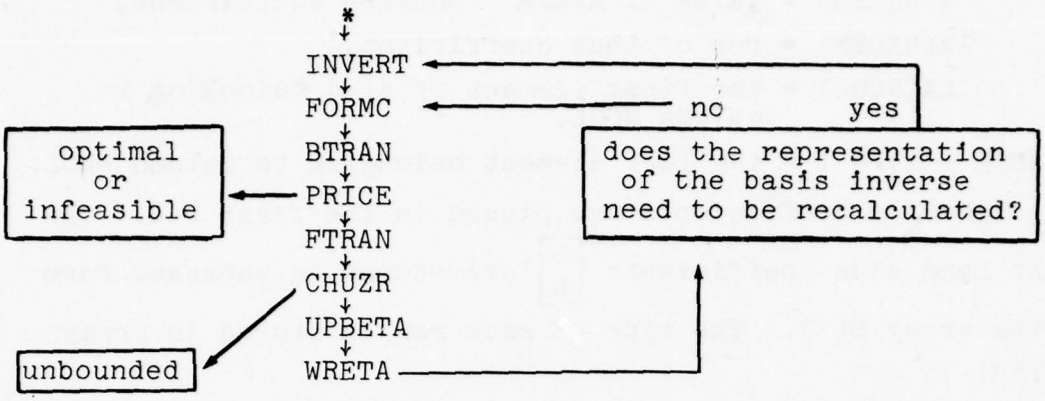
Initially the starting basis is composed of the slack and artificial variables. On subsequent calls of LINPRG, the last basis of the previous problem is used as the starting basis with those variables excluded from the basis by MOGG replaced by the correpsonding slack or artificial. The basic variables are doubly indexed by the arrays JH(·) and KINBAS(·).

JH(ROW) = that basic variable that pivots on row ROW

KINBAS(NCOL) = pivot row of variable NCOL if it is a basic variable; 0 otherwise.

1. Major Subroutines

LINPRG uses 12 subroutines--eight are major, three are bookkeeping, and one prints out the iteration path. The eight major subroutines form the component parts of the simplex cycle with LINPRG linking them together. Each cycle through the following flowchart corresponds to one cycle of the simplex method with a basic/nonbasic variable interchange.



a. INVERT (Invert the Basis)

INVERT starts with the list of basic variables stored in the array JH(·). Using the corresponding coefficients stored in array A(·), it calculates the inverse of the basis (denoted by B) using LU decomposition. The procedure is described in detail in Reference [6].

In general, the matrix of basis coefficients, B, is first decomposed using Gaussian elimination into the product of a lower triangular matrix, L, and an upper triangular mitrix, U:

$$B = LU \quad \text{and} \quad B^{-1} = U^{-1}L^{-1} .$$

Once this is done, a representation of the basis inverse is immediate since the inverse of a triangular matrix is a simple rearrangement of the matrix itself. As an example

$$U^{-1} = \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}^{-1} = \begin{bmatrix} 1/u_{11} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & -u_{12}/u_{22} & 0 \\ 0 & 1/u_{22} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -u_{13}/u_{33} \\ 0 & 1 & -u_{23}/u_{33} \\ 0 & 0 & 1/u_{33} \end{bmatrix}$$

and likewise for L^{-1}.

The LU decomposition of the basis is not unique and one wants to choose that one which (1) minimizes the number of nonzero entries so that storage requirements are reduced and

the number of computation in the BTRAN and FTRAN operations are minimized; and (2), involves division (e.g., $1/u_{11}$, $1/u_{22}$, $1/u_{33}$) by numbers as large as possible to minimize the growth of errors (improve numerical stability). The search for such a decomposition is guided by the tolerance ZTOLPV which will be described in Section 3.

As shown above, the representation of the basis inverse can be written as the product of elementary column matrices E_t (often called eta vectors):

$$B^{-1} = \underbrace{E_t \cdots}_{\| \; U^{-1}} \underbrace{E_2 E_1}_{\| \; L^{-1}}$$

with

$$E_t = \begin{bmatrix} 1 & & \eta_1 & & \\ & \ddots & \vdots & \bigcirc & \\ & & \eta_p & & \\ & \bigcirc & \vdots & \ddots & \\ & & \eta_m & & 1 \end{bmatrix}.$$

The eta vectors are stored in the one-dimensional array E(·). The location of coefficients is maintained by the row index array IE(·) and the eta vector pointer array LE(·).

E(NELEM) = value of NELEMth nonzero coefficient,
IE(NELEM) = row of that coefficient,
LE(ETA) = the first coefficient of E(·) belonging to the eta vector ETA.

b. FORMC (Form the Cost Row)

FORMC checks to see if any variables are at a negative value and, if so, computes the Phase I objective and stores it in work region Y(·). Otherwise, it stores the Phase II objective, Y(1) = 1, in Y(·).

c. BTRAN (Backward Transformation)

BTRAN computes the π vector (multipliers) and stores it in $Y(\cdot)$:

$$Y(\pi) = \underset{\uparrow}{Y} (E_t \ldots E_2 E_1)$$

$\uparrow$ objective passed from FORMC

BTRAN is called the "backward transformation" since it processes the elementary transformation matrices in the reverse order in which they were created.

d. PRICE (Price Out the Nonbasic Variables)

PRICE computes the reduced cost, d_j, for those columns of the coefficient matrix eligible to enter the basis (nonbasic and not excluded by MOGG):

$$d_j = Y(\pi)A(j)$$

(where $A(j)$ is the j^{th} column of the coefficient matrix). PRICE then selects that column which will enter the basis, JCOLP.

e. FTRAN (Forward Transformation)

FTRAN updates the column of coefficients corresponding to the incoming column, JCOLP, and places the result in $Y(\cdot)$:

$$Y = E_t \ldots E_2 E_1 A(\text{JCOLP}) \ .$$

FTRAN(1) is the normal FTRAN described above. FTRAN(2) uses only the elementary matrices associated with the upper triangular factor of B and is used only in the subroutine INVERT. FTRAN is called the "forward transformation" since it uses the elementary transformation matrices in the order in which they were created.

f. CHUZR (Choose That Row Whose Basic Variable Leaves the Basis)

CHUZR finds the pivot row, IROWP, using the ratio tests described in [3].

g. UPBETA (Update the Values of the Current Basic Variables)

UPBETA updates the current basic variable values stored in array X(·) so that they correspond to the new basis.

h. WRETA (Write Eta)

WRETA computes the new eta vector (elementary matrix E_{t+1}) and adds it to the representation of the basis inverse, array E(·):

$$E_{t+1} = \begin{bmatrix} 1 & & \eta_1 & & \\ & \ddots & \vdots & \bigcirc & \\ & & \eta_p & & \\ & \bigcirc & \vdots & \ddots & \\ & & \eta_m & & 1 \end{bmatrix}$$

where

$$\eta_i = \begin{cases} 1/Y(IROWP) & \text{for } i = IROWP \\ -Y(i)/Y(IROWP) & \text{otherwise.} \end{cases}$$

(Actually, the divisions are done only when E_{t+1} is used).

2. Bookkeeping Subroutines

a. SHIFTR (Shift Values in the Work Regions)

LINPRG has two work region arrays, Y(·) and YTEMP(·). Subroutine SHIFTR can shift around the values of any of the

following four arrays:

1	2	3	4
B(·)	X(·)	Y(·)	YTEMP(·)

For example, SHIFTR(1,3) places the values of B(·) into array Y(·), while SHIFTR(4,3) places the values in YTEMP(·) into array Y(·).

b. UNPACK (Unpack a Column of Coefficients from the Constraint Matrix)

Subroutine UNPACK(JCOLP) unpacks the coefficients of column JCOLP and places them in array Y(·).

c. SHFTE (Shift Element of Array E(·))

SHFTE is a bookkeeping subroutine used by INVERT. It is used to manipulate the elementary transformation matrices associated with the upper and lower triangular factors of B.

C. LINPRG OUTPUT

Figure 1 is an example of output generated by LINPRG, most of which was produced by the subroutine ITEROP.

ITCOUNT = iteration number (one cycle of the simplex method is an iteration.

STATUS =
- I — if current basis is infeasible.
- F — if current basis is feasible.
- U — if solution is unbounded.
- blank — if current basis is optimal.

OBJ VALUE = the current value of the objective function (if STATUS is I, OBJ VALUE is the sum of infeasibilities).

VECIN = the nonbasic variable coming into the basis.

VECOUT = the basic variable leaving the basis.

DJ = the adjusted cost of the variable coming into the basis.

NETA = the number of eta vectors which form the current representation of the basis inverse.

```
          0.0
INVERT STATISTICS
  16 NONZ IN BASIS
   0 STRUCTURAL COLUMNS IN BASIS
   0 VECTORS ABOVE BUMP
  16 VECTORS BELOW BUMP
LI    0 NONZ    0 ETAS
UI    0 NONZ    0 ETAS
TOTALS:    0 OFF DIAG NONZ    0 ETAS

ITCOUNT  STATUS    OBJ VALUE          VECIN      VECOUT          DJ          NETA  NELEM  TIME
IROWP=  15  X(I)=      1.00000000  Y(I)=      1.00000000  JM(I)=   15
     1   I          7.25440985       60         15        -6.21997760       0     0     0.00
IROWP=   9  X(I)=      0.00000000  Y(I)=    424.38320429  JM(I)=    9
     2   I          4.00000000       17          9        -1.00000000       1     3     0.00
IROWP=  12  X(I)=      1.00000000  Y(I)=      1.00000000  JM(I)=   12
     3   I          4.00000000       22         12        -1.00000000       2     6     0.00
IROWP=  10  X(I)=      0.00000000  Y(I)=    368.41600000  JM(I)=   10
     4   I          3.00000000       23         10        -1.00000000       3     8     0.00
IROWP=  13  X(I)=      1.00000000  Y(I)=      1.00000000  JM(I)=   13
     5   I          3.00000000       28         13        -1.00000000       4    11     0.00
IROWP=  11  X(I)=      0.00000000  Y(I)=     23.02600000  JM(I)=   11
     6   I          2.00000000       29         11        -1.00000000       5    13     0.00
IROWP=   3  X(I)=       .21448456  Y(I)=       .21448456  JM(I)=    3
     7   I          2.00000000       34          3        -1.00000000       6    16     0.00
IROWP=   2  X(I)=      1.44254871  Y(I)=      1.44254871  JM(I)=    2
     8   I          1.00000000       61          2        -1.00000000       7    18     0.00
IROWP=  16  X(I)=      0.00000000  Y(I)=       .02467556  JM(I)=   16
     9   F         -0.00000000       35         16         -.02467556       8    21     0.00
     9             -0.00000000       35         16         0.00000000       9    25     0.00    STABILITY COUNT =    0
```

Figure 1. LINPRG OUTPUT

NELEM = the number of nonzero elements which form the current representation of the basis inverse.

TIME = 0.00, as the program timer currently is not connected.

IROWP = current pivot row.

X(I) = the adjusted right hand side on row IROWP.

Y(I) = the current pivot element.

STABILITY COUNT will be explained in the next chapter.

Whenever INVERT recalculates the inverse of the basis, it prints those statistics listed in Figure 1 under INVERT STATISTICS. These statistics relate to the LU factorization of the basis and should be of little concern in running normal problems.

D. TOLERANCES AND OTHER CONTROLS

LINPRG uses preset tolerances to reduce the computer running time and accumulated error. These tolerances may need to be adjusted as the code is run on different problems or computers. This section is an attempt to explain what these tolerances do and how they should be adjusted.

Solving large linear programming problems involves adding, subtracting, multiplying, and dividing many numbers. On any digital computer there are round-off errors involved in representing numbers and in using them in operations. For most programs the precision of the computer is such that the accumulated error is negligible. Unfortunately, most linear programming algorithms are designed such that operations are performed on the results of operations and, when this is done often enough, the accumulated error can grow to significant levels even on precise machines. LINPRG uses the revised simplex method. It carries along a representation of the inverse of the current basis, B^{-1}, which is a product of past computations and has with it an accumulated error.

$$B^{-1} = E_t E_{t-1} \cdots E_1 .$$

(E_t is the most recently added elementary column matrix.) Each time the basis is changed, a new elementary matrix is added to B^{-1} and with it possibly some error. At some point the errors may get out of hand and B^{-1} will no longer be a good approximation to the inverse of the basis. Since the algorithm is vitally dependent on B^{-1}, it can then wander off and do ridiculous things.

The accumulated error is a function of the number of computations and the size of the round-off error involved in those computations. In general, the tolerances allow the code to neglect insignificant numbers and, when choice is possible, to perform those computations with the smallest round-off error.

1. Tolerances Used in LINPRG

ZTOLZE is the zero tolerance used throughout the program. Its purpose is to zero out any "background noise" and thereby reduce storage requirements and the number of computations. It should be slightly larger than the precision of the machine--for our machine this is $2^{-60} \approx 10^{-18}$. If set too low, storage requirements will be significantly increased. If set too high, "good" numbers will be thrown away and accuracy reduced.

ZTOLCR is the pivot tolerance used in CHUZR. CHUZR selects the old basic variable that leaves the basis and thereby the divisor (called the pivot element) which is adjoined to the representation of the new inverse. That divisor must have a magnitude greater than ZTOLCR. This keeps the algorithm from dividing by small numbers and thus creating large ones which would increase the chance of round-off error in subsequent computations. If ZTOLCR is set too high, the algorithm may go to an infeasible basis from a feasible one;

it may even terminate with an unbounded solution when this should not be the case. If ZTOLCR is set too low, errors will grow rapidly when the algorithm is run on problems which are inherently unstable.

ZTOLPV is the absolute pivot tolerance used in the reinversion subroutine INVERT. It functions in essentially the same way as ZTOLCR. Increasing ZTOLPV increases the minimum size of the pivot elements in the new representation of the inverse and thereby increases the stability. Decreasing ZTOLPV will allow the representation to have fewer nonzero elements and will decrease the number of computations required by the algorithm. The tests of Reference [6] suggest the following value:

$$\text{ZTOLPV} = (10^2 \cdot \max|a_{ij}|)^{-1}$$

where a_{ij} are the coefficients of the current basis.

ZTOLPV and ZTOLCR are related in that the revised simplex part of the code hands over a basis to INVERT which is nonsingular with respect to the pivot tolerance ZTOLCR. If ZTOLPV is greater than ZTOLCR, then the reinversion subroutine INVERT may find that, from its viewpoint, the basis is singular and can not be inverted. Setting ZTOLPV less than or equal to ZTOLCR will avoid this problem.

ZTCOST regulates the tightness of the terminating test. If the minimum adjusted cost is within ZTCOST of zero, then the algorithm terminates. It should be noted that this tolerance does not affect stability. If it is too large, the solution returned upon termination may not be optimal. If it is too small, the computer time will become excessive as background noise dominates.

2. Reinversion

No matter how well the tolerances are set, at some point the accumulated error will grow to significant levels. When this happens, the representation of the basis inverse should be recalculated. This is done by the subroutine INVERT. (Reinverting is expensive in terms of time and should be done only when necessary. To identify when it becomes necessary is, in itself, a problem.) The accumulated error could be calculated directly by computing $||B^{-1}B - I||_{\infty}$. Unfortunately, this would take about as much time as reinverting the basis itself, as the FTRAN subroutine would have to be called for each column in the basis in calculating $B^{-1}B$.

An indirect measure of the accumulated error which takes relatively little time to compute is the STABILITY COUNT, which appears in MOGG output for every call of LINPRG. It is computed as follows. In the subroutine PRICE the adjusted cost, d_j, is calculated for each nonbasic variable, and the most negative is then chosen to enter the basis.

$$\text{adjusted cost} = d_j\,(\text{BTRAN}) = c_j - c_b B^{-1} A(j) \ .$$

c_j is the cost for variable j; c_b is the vector of the basic costs; and $A(j)$ is column j of the coefficient matrix. This is done by using the subroutine BTRAN to compute the multipliers.

$$\pi = c_b B^{-1} = c_b \, E_t \cdots E_1 \ .$$

This is done once and π is then applied iteratively to each $A(j)$ to compute the adjusted cost values in PRICE. π is calculated by multiplying c_b and E_t and then the result by E_{t-1} and so on. Notice that the elementary matrices are multiplied from left to right, thus BTRAN is called the

"backward transformation." Once PRICE has selected the nonbasic variable to enter the basis, CHUZR selects the variable to leave the basis. It needs the adjusted coefficients of the incoming variable

$$\bar{A}(j) = B^{-1}A(j) = E_t \ldots E_1 A(j) .$$

This computation is done by FTRAN which multiplies the elementary matrices from right to left; thus FTRAN is called the "forward transformation." Notice that with just one vector multiplication, d_j can be recalculated

$$d_j(\text{FTRAN}) = c_j - c_b\bar{A}(j) .$$

Theoretically, matrix multiplication is associative and thus d_j(BTRAN) should equal d_j(FTRAN). The only way they can be unequal is if the "stability" of the representation of the basis inverse has degraded to the point where the accumulated round-off error generated in using it becomes significant. The STABILITY COUNT is the number of times

$$|d_j(\text{BTRAN}) - d_j(\text{FTRAN})| > \text{ZTOLZE}$$

occurs. Notice the rather subtle point that the STABILITY COUNT says nothing directly about whether the current representation of the basis is accurate. What it does say is that if you multiply a vector by that representation the result will be affected significantly by round-off errors. Since the elementary matrix (which gets added to the representation of the inverse at each iteration) is a product of such a calculation, it is likely that it will also be in error.

It has been found experimentally that FTRAN accumulates significantly less round-off error than BTRAN. For this reason, when mismatches occur, d_j(FTRAN) is probably more accurate than d_j(BTRAN). If both d_j(BTRAN) and d_j(FTRAN) are negative, then the entering nonbasic variable should decrease the objective.

If d_j(FTRAN) turns out to be positive, then we are probably going in the wrong direction. At this point, LINPRG returns to PRICE to try again. If d_j(FTRAN) turns out to be positive once again, then the basis inverse is recalculated by calling INVERT.

There are two other reasons for reinverting the basis. At each iteration of the simplex method, an elementary matrix gets added to the representation of the inverse. At some point the storage space will be exceeded and one must recalculate the basis inverse representation. The storage space is especially critical for all-in-core codes like LINPRG. The other reason for reinverting is to improve the running time. As the representation of the inverse gets larger, it takes more computer time to use it. At some point it will become advantageous to expend time reinverting the basis to reduce the size of the representation of the inverse and, thus, the time required to use it in the simplex method.

Figure 2 roughly illustrates how the computer time that it takes to complete an iteration of the simplex method will increase as the size of the representation of the inverse increases. Figure 3 illustrates that the time it takes to reinvert the basis will generally remain constant once the algorithm reaches Phase II. The key question, of course, is when should the basis be reinverted to improve the running time. The best solution is to access the program timer and keep track of the time it takes for each iteration and then reinvert according to some rule, such as:

Reinvert at iteration I if

$$1/2(\text{ITIM}(I) - \text{ITIM}(0)) \times I \geq \text{INVTIM} .$$

LINPRG does not use such a rule, since program timers are machine dependent and make it difficult to transfer the code from machine to machine. Currently, LINPRG reinverts the

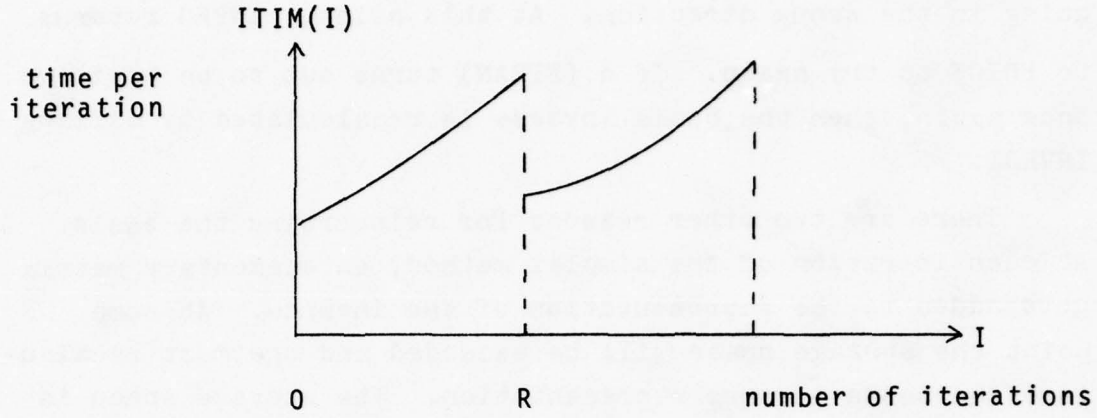


Figure 2. COMPUTER TIME PER ITERATION

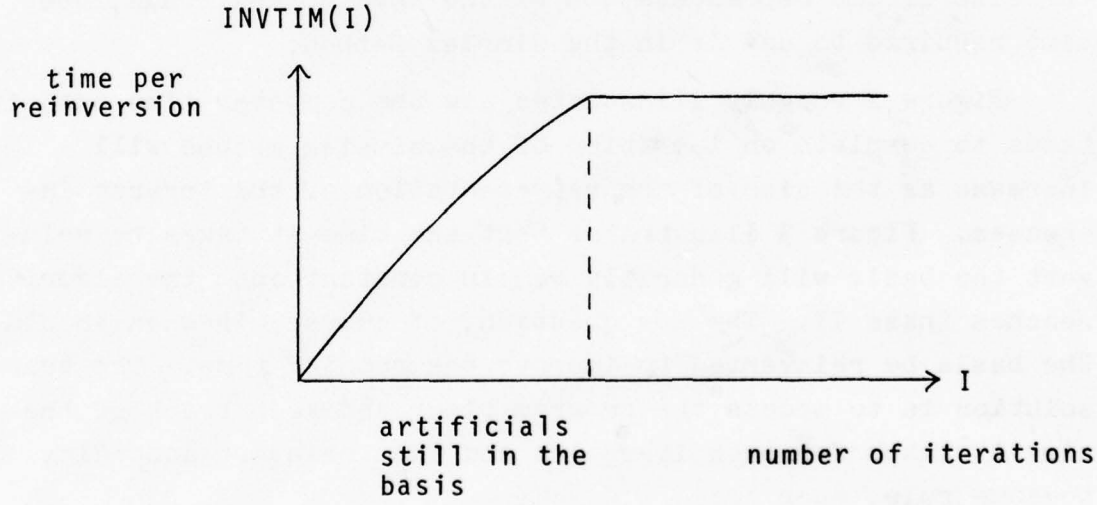


Figure 3. COMPUTER TIME PER REINVERSION

basis at least every 50 iterations. This appears to be a reasonable approximation to a more sophisticated rule such as described above.

3. Tolerance Values

Indications of tolerance problems are:

1. STABILITY COUNT greater than zero,
2. A large change in the objective value after reinversion,
3. Algorithm goes from a feasible to an infeasible basis,
4. Algorithm takes more time than it should or goes unbounded when it should not.

The principal point to remember is that if the algorithm does not behave as it should, the tolerances should be adjusted. If one wants increased confidence in the solution, check to see that the stability count is low or at least that mismatches do not occur near termination. If mismatches occur, adjust the tolerances and run again. If degeneracy occurs, perturbing the right hand side by a small amount should help (this is called "epsilon perturbation").

The following is a list of values for the tolerances. The Orchard-Hays column are values suggested by Reference [3]. The LINPRG column are those values used in LINPRG. The range values are the author's estimates of reasonable upper and lower values for the tolerances. It should be emphasized that the arguments for setting tolerance values are generally heuristic and best values will vary from problem to problem.

	Orchard-Hays	LINPRG	Range
ZTOLZE	10^{-12}	10^{-10}	$10^{-18} \leq 10^{-8}$
ZTCOST	--	10^{-10}	$0.0 \leq 10^{-2}$
ZTOLPV	10^{-12}	10^{-6}	$10^{-10} \leq 10^{-3}$
ZTOLCR	10^{-5}	10^{-4}	$10^{-10} \leq 10^{-3}$
INVFRQ	--	50	$10 \leq 100$

REFERENCES

[1] Dantzig, G.B. and W. Orchard-Hays, "The Product Form of the Inverse in the Simplex Method," Match. Tables Aids. Comput. 8, pp. 64-7, 1954.

[2] McCoy, P.F. and J.A. Tomlin, "Some Experiments on the Accuracy of Three Methods of Updating the Inverse in the Simplex Method," Stanford Univ. Systems Optimization Laboratory, Technical Report 74-21, December 1974.

[3] Orchard-Hays, William, *Advanced Linear-Programming Computing Techniques*, New York: McGraw-Hill, 1968.

[4] Tomlin, J.A., "Maintaining a Sparse Inverse in the Simplex Method," IBM Journal of Res. and Dev. 16, pp. 415-23, 1972.

[5] Tomlin, J.A., "Modifying Triangular Factors of the Basis in the Simplex Method," Sparse Matrices and Their Applications (Rose and Willoughby, eds), New York: Plenum Press, pp. 77-85, 1972.

[6] Tomlin, J.A., "Pivoting for Sparsity and Size in Linear Programming Inversion Routines," J. Inst. Maths. Applics. 10, pp. 289-95, 1972.

[7] Wilkinson, J.H., *Rounding Errors in Algebraic Processes*, New York: Prentice Hall, 1963.

APPENDIX C

MOGG LISTING

```
      PROGRAM MOGG(INPUT,OUTPUT,TAPE6=OUTPUT)
C     *****MAXVAR, THE MAXIMUM NUMBER OF ORIGINAL VARIABLES, DIMENSIONS
C          KLO,KRO,KL,KR,XCODED,XBEST,VARNAM
C     *****MAXCUT, THE MAXIMUM NUMBER OF INTERNAL VARIABLES DIMENSIONS
C          W,CUTS
C     *****LSTMAX, THE MAXIMUM LIST SIZE DIMENSIONS ZLSINO,ZLSTPA
C          LSTKL, LSTKR,ZLSLB ,IBRVR,FLAG
C     *****MAXROW=MAXIMUM NUMBER OF ROWS (INC. OBJECTIVE FUNCTION),
C          DIMENSIONS ICHK
C     *****MAXA DIMENSIONS A,IA
      COMMON/FIRST/KLO(100),KRO(100),KL(100),KR(100),XCODED(100),XBEST
     1(100),W(1100),CUTS(1100),ZLSTNO(700),ZLSTPA(700), LSTKL(700),
     2 LSTKR(700),ZLSLB(700),IBRVR(700),FLAG(700),KBL(3),KBR(3),
     3VARNAM(100),PROBNA(8),MAXVAR,MAXCUT, LSTMAX,MAXROW,MAXA,
     4 NMROWS,NUMVAR,ICHK(100),VAL,LFLG
      COMMON/WORK1/ B(350),X(350),Y(350),YTEMP(350),A(5000),E(5000),
     1  IA(5000),IE(5000),LA(1302),LE(2002),ICNAM(1302,2),KINBAS(1302),
     2  JH(350),ISTYPE(350),NAME(20),NTEMP(20),CMIN,COND,ERMAX,IFFEZ,
     3  INVFRQ,IOBJ,IROWP,IICH,ITCHA,ITCNT,ITRFRQ,IVIN,IVOUT,JCOLP,KINP,
     4  XSTAT,NROW,NCOL,NELEM,NETA,NLELEM,NLETA,NUELEM,NGETA,NUELEM,
     5  NUETA,SUMINF,K3
      COMMON/BLOCK/ ZTOLZE,ZTOLPV,ZTCOST,NRMAX,NTMAX,NEMAX,QRO,QMA,QBA,
     1  QFI,QEO,QBL,QPL,QMI,QA,QB,QC,QE,QF,QG,QH,QI,QL,QM,QN,QO,QR,QU,QZ
      DATA  IOBJ,INVFRQ,ITRFRQ/1,50,1000/
      DATA ZTOLZE,ZTOLPV,ZTCOST/1.E-10,1.E-6,1.E-10/
      DATA NRMAX,NTMAX,NEMAX/350,2000,5000/
      DATA QRO,QMA,QBA,QFI,QEO,QBL,QPL,QMI/4HROW ,4HMATR,4HBASI,
     1  4HFIRS,4HEOF ,4H     ,4H +  ,4H -  /
      DATA QA,QB,QC,QE,QF,QG,QH,QI,QL,QM,QN,QO,QR,QU,QZ/4HA   ,4HB   ,
     1  4HC   ,4HE   ,4HF   ,4HG   ,4HH   ,4HI   ,4HL   ,4HM   ,4HN   ,
     2  4HO   ,4HR   ,4HU   ,4H Z  /
      DATA BUBTOL,FEASTL,DIFFTO,DONTOL/1.E-10,1.E-6,1.E-6,1.E-6/
      DATA CUTTOL/1.E-8/
      MAXVAR=100
      MAXCUT=1100
      LSTMAX=700
      MAXROW=100
      MAXA=5000
      WMM=5HMANU.
      BUB=1.E70
C     *****THIS SECTION READS IN DATA
      PRINT 5
 5    FORMAT(61H1PROGRAM MOGG--FINDS GLOBAL SOLUTIONS TO APPROXIMATE PRO
     1BLEMS)
      READ 10,NMROWS,NUMVAR,MAXLP,KBUB,IXPRIN,K1,K2,K3,K4,K5
 10   FORMAT(10I5)
      IF(KBUB.NE.0)READ 12,BUB
 12   FORMAT(F10.6)
      PRINT 15
 15   FORMAT(1H0,19HPROBLEM INFORMATION,/)
      PRINT 20,NMROWS
 20   FORMAT(1H ,20X,I10,4HROWS)
      PRINT 25,NUMVAR
 25   FORMAT(1H ,20X,I10,9HVARIABLES)
      PRINT 30,MAXLP
 30   FORMAT(1H ,20X,I10,5X,26HLP PROBLEMS WILL BE SOLVED)
      IF(KBUB.NE.0)PRINT 35,BUB
```

```
 35   FORMAT(1H ,20X,22HUSER-SUPPLIED BUB TS=,F10.6)
      IF(IXPRIN.NE.0)PRINT40
 40   FORMAT(1H ,20X,59HTHE USER REQUESTS THAT ALL FEASIBLE POINTS FOUND
     1 BE PRINTED)
      IF(K1.NE.0)PRINT45
 45   FORMAT(1H ,20X,50HTHE USER REQUESTS THAT ALL LP SOLUTIONS BE PRINT
     1ED)
      IF(K2.NE.0)PRINT 50
 50   FORMAT(1H ,20X,44HTHE USER REQUESTS THAT THE MATRIX BE PRINTED)
      IF(K3.NE.0)PRINT 51
 51   FORMAT(1H ,20X,43HTHE USER REQUESTS LP INFORMATION BE PRINTED)
      IF(K4.NE.0)PRINT 52
 52   FORMAT(1H ,20X,66HTHE USER REQUESTS THAT THE ENTIRE LIST BE PRINTE
     1D AFTER EACH STAGE)
      IF(K5.NE.0)PRINT 53
 53   FORMAT(1H ,20X,43HTHE USER REQUESTS THAT THE MATRIX BE SCALED)
      READ 55,(ISTYPE(I),I=1,NMROWS)
 55   FORMAT(40I2)
      PRINT60
 60   FORMAT(1H0,10HROW TYPE--)
      PRINT 65,(ISTYPE(I),I=1,NMROWS)
 65   FORMAT(1H , 40I2)
      READ 70,(ICHK(I),I=1,NMROWS)
 70   FORMAT(80I1)
      PRINT 75
 75   FORMAT(1H0,17HCONVEXITY FLAGS--,/)
      PRINT 80,(ICHK(I),I=1,NMROWS)
 80   FORMAT(1H ,80I1)
C     *****NOW SET UP CUTS VECTOR, KLO, AND KRO--
      PRINT 90
 90   FORMAT(1H0,27HVARIABLE CARDS REPRODUCED--,/)
           DO 100 I=1,NUMVAR
           READ 105,NOVAR,NOINC,WORD
105        FORMAT(I5,I5,A5)
      IF(NOVAR.NE.I)CALL ERR(1)
           PRINT 110,NOVAR,NOINC,WORD
110        FORMAT(1H ,I5,I5,A5)
           IF(NOINC.EQ.0)115,120
115        IF(I.EQ.1)116,117
116        KLO(I)=1
           KRO(I)=1
           GO TO 100
117        IX=KRO(I-1)+1
      IF(IX.GT.MAXCUT)CALL ERR(2)
           KLO(I)=KRO(I-1)+1
           KRO(I)=KLO(I)
           GO TO 100
120        IF(I.EQ.1)122,124
122        KLO(I)=1
           GO TO 126
124        IX=KRO(I-1)+1
      IF(IX.GT.MAXCUT)CALL ERR(2)
           KLO(I)=KRO(I-1)+1
126   IF((KLO(I)+NOINC).GT.MAXCUT)CALL ERR(2)
           KRO(I)=KLO(I)+NOINC
           IF(WORD.EQ.WMM)GO TO 145
           I1=KLO(I)
           I2=KRO(I)
```

```
      READ 130,CUTS(I1),CUTS(I2)
130   FORMAT(2F10.6)
      PRINT 135,CUTS(I1),CUTS(I2)
135   FORMAT(1H ,2G10.4)
      IF((I2-I1).EQ.1)GO TO 100
           IX=I2-I1-1
           DO 140 J=1,IX
140        CUTS(I1+J)=CUTS(I1)+J*(CUTS(I2)-CUTS(I1))/NOINC
      GO TO 100
145   CONTINUE
C     *****HERE IF WE ARE TO READ IN CUTS MANUALLY
      IW=KLO(I)
      IZ=KRO(I)
      READ 150,(CUTS(J),J=IW,IZ)
150   FORMAT(8F10.6)
      PRINT 155,(CUTS(J),J=IW,IZ)
155   FORMAT(1H ,8G12.4)
100   CONTINUE
C     *****WE HAVE COMPLETED READING BOUNDS AND CUTS
      PRINT 160
160   FORMAT(1H0,24HRHS CARD(S) REPRODUCED--,/)
      READ 165,(B(I),I=1,NMROWS)
165   FORMAT(8F10.6)
      PRINT 170,(B(I),I=1,NMROWS)
170   FORMAT(1H ,8G12.4)
C
C        SET NROW,B(.),ISTYPE(.)
C
      NROW=NMROWS
      DO 9000 JJ=1,NUMVAR
      J1=KLO(JJ)
      J2=KRO(JJ)
      IF (J1.EQ.J2)  GO TO 9000
      NROW=NROW+1
9000  CONTINUE
C
      I1=NMROWS+1
      DO 9010 I=I1,NROW
      B(I)=1.
      ISTYPE(I)=-1
9010  CONTINUE
C
C        ADD SLACKS TO COEFFICIENT MATRIX
C
      NELEM=0
      NCOL=0
      DO 9100 I=1,NROW
      NELEM=NELEM+1
      NCOL=NCOL+1
      IA(NELEM)=I
      A(NELEM)=1.
      JA(NCOL)=NELEM
9100  CONTINUE
      JA(NCOL+1)=NELEM+1
C
C
C        FILL IN COEFFICIENT MATRIX
C
```

```
      NCOGUB=1
      DO 9400 JJ=1,NUMVAR
      J1=KLO(JJ)
      J2=KRO(JJ)
      IF (J1.LT.J2)  GO TO 9300
C
      READ 9250, (YTEMP(I),I=1,NMROWS)
 9250 FORMAT(8F10.6)
C
      DO 9270 I=1,NMROWS
      YTEMP1=YTEMP(I)
      IF (ABS(YTEMP1).LE.ZTOLZE)  GO TO 9270
      NELEM=NELEM+1
      IA(NELEM)=I
      A(NELEM)=YTEMP1
 9270 CONTINUE
      NCOL=NCOL+1
      LA(NCOL+1)=NELEM+1
      GO TO 9400
C
 9300 DO 9390 J=J1,J2
C
      DO 9380 I=1,NMROWS
      CALL GETPHI(I,JJ,CUTS(J),ATEMP)
      IF (ABS(ATEMP).LE.ZTOLZE)  GO TO 9380
      NELEM=NELEM+1
      IA(NELEM)=I
      A(NELEM)=ATEMP
 9380 CONTINUE
      NELEM=NELEM+1
      IA(NELEM)=NMROWS+NCOGUB
      A(NELEM)=1.
      NCOL=NCOL+1
      LA(NCOL+1)=NELEM+1
 9390 CONTINUE
      NCOGUB=NCOGUB+1
 9400 CONTINUE
      IF(NELEM.GT.MAXA)CALL ERR(3)
      READ 200,(VARNAM(I),I=1,NUMVAR)
  200 FORMAT(16A5)
      READ 250,(PROBNA(I),I=1,8)
  250 FORMAT(8A10)
C     *****DONE READING IN DATA
      PRINT 275
  275 FORMAT(1H0,71HFOR YOUR INFORMATION           VARIABLE NUMBER
     1   KLO           KRO,/)
           DO 280I=1,NUMVAR
           PRINT 285,I,KLO(I),KRO(I)
  285      FORMAT(1H ,35X,I5,14X,I5,8X,I5)
  280      CONTINUE
C     ******SET UP STARTING BASIS
           DO 9200 I=1,NROW
           JH(I)=I
 9200      KINBAS(I)=I
      IF(K5.EQ.1)CALL SCAIL
      IF(K2.NE.0)360,370
  360 PRINT 361
  361 FORMAT(1H0,55HPACKED MATRIX BY COLUMNS, ROW NUMBER BELOW EACH ELEM
```

```
     1ENT)
      IXI=NELEM/11+1
      DO 365 I=1,IXI
      KK=(I-1)*11+1
      IF(KK.GT.NELEM)GO TO 370
      K=KK+10
      IF(K.GT.NELEM)K=NELEM
      PRINT 366,(A(J),J=KK,K)
      PRINT 367,(IA(J),J=KK,K)
  366 FORMAT(1H0,11G12.4)
  367  FORMAT(1H ,11I12)
  365 CONTINUE
  370 CONTINUE
      PRINT 290
  290 FORMAT(1H1,19HSTARTING TO ITERATE)
      PRINT 300
  300 FORMAT(1H0,4X,13HSTAGE,PROBLEM,7X,11HLOWER BOUND,9X,11HUPPER BOUND
     1,6X,18HBRANCHING VARIABLE,//)
C     *****READY TO START LOOP
C     *****READY TO START LOOP
C     *****READY TO START LOOP
C     *****READY TO START LOOP
          DO 980 I=1,LSTMAX
          IBRVR(I)=0
  980     ZLSLB(I)=1.E80
      PARENT=0.0
      LSTKR(1)=0
      LSTKL(1)=0
      STGPRB=0.0
      LSTNUM=0
      NLP=0
      IBRPAR=0
      NOLFT=1
          DO 990 I=1,NUMVAR
          KL(I)=KLO(I)
  990     KR(I)=KRO(I)
 1000 PRINT 1005,STGPRB
 1005 FORMAT(1H0,8X,F6.1)
      NOLFT=NOLFT-1
      NLP=NLP+1
      IF(NLP.GT.MAXLP)CALL ERR(4)
C
      JK=KRO(NUMVAR)
      DO 6300 J=1,JK
C
 6300 W(J)=0.0
      CALL LINPRG
      IF(K1.EQ.1)CALL LPPR(W)
      IF(LFLG.NE.1)GO TO 1010
      IF(NLP.EQ.1)CALL ERR(5)
      PRINT 1008
 1008 FORMAT(1H*,24X,11HLP. INFEAS.)
      GO TO 5000
 1010 IF(VAL-BUBTOL.LE.BUB)GO TO 1020
      PRINT 1015
 1015 FORMAT(1H*,25X,9HLB GT BUB)
      GO TO 5000
C     *****PUT THIS PROBLEM ON THE LIST, FIND THE NEXT EMPTY SPOT
```

```
 1020 IF (LSTNUM.GT.LSTMAX)CALL ERR(6)
      PRINT 1025,VAL
 1025 FORMAT(1H+,25X,G12.4)
      LSTNUM=LSTNUM+1
      ZLSTO(LSTNUM)=STGPRB
      ZLSTPA(LSTNUM)=PARENT
      IF(IBRPAR.NE.0)LSTKL(LSTNUM)=KL(IBRD,R)
      IF(IBRPAR.NE.0)LSTKR(LSTNUM)=KR(IBRD,R)
      ZLSLB(LSTNUM)=VAL
C     *****NOT DONE WITH LIST YET
C     *****CREATE XCODED, COMPACTIFIED VERSION OF X--ELEMENTS OF XCODED
C     *****ARE X, IF A CORRESPONDS TO A LINEAR VARIABLE, OR ELSE THE
C     *****MEAN OF GUBBED VARIABLES
           DO 1030 I=1,NUMVAR
           IF(KLO(I).NE.KRO(I))GO TO 1040
           II=KLO(I)
           XCODED(I)=W(II)
           GO TO 1030
 1040      XCODED(I)=0.0
                IW=KLO(I)
                IZ=KRO(I)
                DO 1050 II=IW,IZ
 1050           XCODED(I)=XCODED(I)+W(II)*CUTS(II)
 1030      CONTINUE
C     *****XCODED CREATED, CHECK IT
           DO 1060 I=1,NUMVAR
           IF(KLO(I).EQ.KRO(I))GO TO 1060
           IL=KLO(I)
            IR=KRO(I)
           XXL=CUTS(IL)-CUTTOL
           XXR=CUTS(IR)+CUTTOL
      IF(XCODED(I).LT.XXL)CALL ERR(7)
      IF(XCODED(I).GT.XXR)CALL ERR(7)
 1060 CONTINUE
C     *****NOW WE WILL TRY TO FIND AN UPPER BOUND AND ALSO A
C     *****BRANCHING VARIABLE
      IFEAS=1
      DIFFMA=0.0
      IBRVR(LSTNUM)=1
      DO 8 I=1,NTO
           DO 2000 IROW=1,NMROWS
           IF(ICHK(IROW).EQ.1)GO TO 2000
           ROWVAL=0.0
                DO 2010 I=1,NUMVAR
C     *****SET INDEX AND FRAC
      IW=KLO(I)
      IZ=KRO(I)
      IF(IW.EQ.IZ)7005,7010
 7005 INDEX=-1
      FRAC=0.0
      GO TO 7900
 7010 IM=IZ-1
      DO 7015 IJ=IW,IM
      IK=IJ+1
      IF(CUTS(IJ).LE.XCODED(I).AND.CUTS(IK).GT.XCODED(I))GO TO 7025
 7015 CONTINUE
      XXX=CUTS(IW)-XCODED(I)
      XXX=ABS(XXX)
```

```
      IF(XXX.LT.CUTTOL)GO TO 7016
      XXX=XCODED(I)-CUTS(IZ)
      XXX=ABS(XXX)
      IF(XXX.LT.CUTTOL)GO TO 7017
      CALL ERR(7)
 7025 FRAC=(XCODED(I)-CUTS(IJ))/(CUTS(IK)-CUTS(IJ))
      INDEX=IJ
      GO TO 7900
 7016 INDEX=IW
      FRAC=0.0
      GO TO 7900
 7017 INDEX=IZ
      FRAC=0.0
 7900 CONTINUE
                DIFF=0.0
                     DO 2020 IJ=IW,IZ
                     II=IJ+NROW
                     INDCOL=LA(II)
                     INDNXT=LA(II+1)-1
                          DO 2030 III=INDCOL,INDNXT
                          IF(IA(III).NE.IROW)GO TO 2030
                          IF(INDEX.NE.-1)DIFF=DIFF-W(IJ)*A(III)
                          IF(INDEX.EQ.IJ)DIFF=DIFF+(1.-FRAC)*A(III)
                          IF(INDEX.EQ.-1)ROWVAL=ROWVAL+XCODED(I)*A(III)
                          IF(INDEX.EQ.IJ)ROWVAL=ROWVAL+(1.-FRAC)*A(III)
                          IF((INDEX+1).EQ.IJ)DIFF=DIFF+FRAC*A(III)
                          IF((INDEX+1).EQ.IJ)ROWVAL=ROWVAL+FRAC*A(III)
 2030                     CONTINUE
 2020                CONTINUE
      IF(ISTYPE(IROW).EQ.-1)DIFF=ABS(DIFF)
                IF(DIFF.LT.DIFFMA+DIFFTO)GO TO 2010
                DIFFMA=DIFF
      FLAG(LSTNUM)=INDEX
      FLAG(LSTNUM)=FLAG(LSTNUM)+FRAC
                IBRVR(LSTNUM)=I
 2010           CONTINUE
           IF(ISTYPE(IROW))2040,2050,2060
 2050      UB=ROWVAL
           GO TO 2000
 2060 IF(B(IROW))2061,2062,2063
 2061 IF(ROWVAL.LE.(B(IROW)*(1.-FEASTL)))GO TO 2000
      IFEAS=1
      GO TO 2000
 2062 IF(ROWVAL.LE.FEASTL)GO TO 2000
      IFEAS=1
      GO TO 2000
 2063 IF(ROWVAL.LE.(B(IROW)*(1.+FEASTL)))GO TO 2000
      IFEAS=1
      GO TO 2000
 2040      IF(ABS(B(IROW)).EQ.0.0)GO TO 2070
           XXX=1.-ABS(ROWVAL/B(IROW))
           IF(ABS(XXX).LT.FEASTL)GO TO 2000
           IFEAS=1
           GO TO 2000
 2070      IF(ABS(ROWVAL).LE.FEASTL)GO TO  2000
           IFEAS=1
 2000      CONTINUE
C     *****DONE--WE HAVE PICKED A BRANCHING VARIABLE AND STORED IT ON
```

```
C       *****THE LIST WHILE TESTING  FOR FEASIBILITY
        IF(IFEAS.NE.1)GO TO 3000
        PRINT 2005,IBRVR(LSTNUM)
 2005   FORMAT(1H+,49X,4HNONE,16X,I6)
        GO TO 5000
 3000   PRINT 3005,UB,IBRVR(LSTNUM)
 3005   FORMAT(1H+,45X,G12.4,12X,I6)
        IF(IXPRIN.EQ.1)CALL XPRINT(XCODED)
        IF(UB.LT.BUB)GO TO 3010
        GO TO 5000
 3010   BUB=UB
            DO 3020 I=1,NUMVAR
C       *****NOW BEGIN BRANCHING PROCEDURE
 3020       XBEST(I)=XCODED(I)
 5000   IF(NOLFT.EQ.0)GO TO 5050
C       *****SOLVE NEXT PROBLEM IN THIS STAGE
        KL(IBRPAR)=KBL(NOLFT)
        KR(IBRPAR)=KBR(NOLFT)
        STGPRB=STGPRB+.1
        GO TO 1000
C       *****WE ARRIVE HERE IF WE ARE DONE WITH A STAGE
 5050   NUM=0
        BLB=1.E70
            DO 5060 I=1,LSTNUM
            IF(ZLSLB(I).GE.BLB)GO TO 5060
            NUM=I
            BLB=ZLSLB(I)
 5060       CONTINUE
        IF(BLB.GE.1.E70)CALL ERR(8)
        IF(BLB.GE.BUB-DONTOL)GO TO 8000
C       *****NUM IS THE ENTRY ON THE LIST ON WHICH WE ARE TO BRANCH
 5063   FORMAT(1H0,20HDONE WITH THIS STAGE)
        IF(IBRVR(NUM).NE.0)GO TO 5064
        CALL ERR(9)
 5064   PRINT 5063
        PRINT 5065,BLB,BUB,ZLSLNO(NUM),IBRVR(NUM)
 5065   FORMAT(1H ,6HBLB=  ,G12.4,8H, BUB=  ,G12.4,29H, BRANCHING ON PROBL
       1EM,FE.1,17H, VARIABLE NUMBER,I6)
        IF(K4.EQ.1)PRINT 50651
50651   FORMAT(1H0,32H**********PRESENT STATUS OF LIST)
        IF(K4.EQ.1)PRINT 50652
50652   FORMAT(1H0,6HPROBNO,5X,6HPARENT,5X,6HLISTKL,5X,6HLISTKR,5X,
       111HLOWER BOUND,5X,12HBRANCH. VAR.,8X,4HFLAG,/)
        IF(K4.EQ.1)50653,50657
50653   DO 50009 I=1,LSTNUM
        PRINT 50001,ZLSLNO(I),ZLSTPA(I), LSTKL(I), LSTKR(I)
50001   FORMAT(1H ,F6.1,5X,F6.1,5X,I5,6X,I5)
        IF(ZLSLB(I).GE.1.E70)50002,50004
50002   PRINT 50003,IBRVR(I),FLAG(I)
50003   FORMAT(1H+,48X,3HOFF,4X,9X,I4,9X,F10.3)
        GO TO 50009
50004   PRINT 50005,ZLSLB(I),IBRVR(I),FLAG(I)
50005   FORMAT(1H+,44X,G11.4,9X,I4,9X,F10.3)
50009   CONTINUE
        PRINT 50656
50656   FORMAT(1H0)
50657   CONTINUE
        PARENT=ZLSLNO(NUM)
```

```
      IBRPAR=IBRVR(NUM)
C     *****NOW WE KNOW THE PARENT AND BRANCHING VARIABLE FOR THE NEXT
C     *****STAGE--SET UP 2 OR 3 NEW PROBLEMS
C     *****FILL KL AND KR VECTORS
            DO 5070 I=1,NUMVAR
            KR(I)=KRO(I)
 5070       KL(I)=KLO(I)
      ZNBACK=ZLSTNO(NUM)
            DO 5080 I=1,NUM
            II=NUM-I+1
            IF(ZNBACK.EQ.0.0)GO TO 5110
            IF(ZLSTNO(II).NE.ZNBACK)GO TO 5080
                  DO 5090 IK=1,NUM
                  IF(ZLSTNO(IK).EQ.ZLSTPA(II))GO TO 5095
 5090             CONTINUE
 5095       III=IBRVR(IK)
            IF(.NOT.((LSTKL(II).GE.KL(III)).AND.(LSTKR(II).LE.KR(III))))
     1      GO TO 5100
            KL(III)=LSTKL(II)
            KR(III)=LSTKR(II)
 5100       ZNBACK=ZLSTPA(II)
 5080       CONTINUE
 5110 CONTINUE
C     *****NOW HAVE TO DIVIDE UP THE K-SET FOR THE BRANCHING VARIABLE
C     *****BUT FIRST, REMOVE THE PARENT PROBLEM FROM THE LIST
      ZLSLB(NUM)=1.E70
C     *****SET UP TWO OR THREE PROBLEMS
      IF((FLAG(NUM).LT.KL(IBRPAR)).OR.(FLAG(NUM).GT.KR(IBRPAR)))
     1CALL ERR(11)
C     *****CHECK TO SEE IF FLAG PRECISELY EQUALS SOME CUT
            IW=KL(IBRPAR)
            IZ=KR(IBRPAR)
            DO 5120 J=IW,IZ
            ZJ=J
            XXX=ZJ-FLAG(NUM)
            XXX=ABS(XXX)
            IF(XXX.LE.CUTTOL)GO TO 5130
 5120       CONTINUE
      IX=FLAG(NUM)
      IF(IX.EQ.KL(IBRPAR))GO TO 5140
      KBL(1)=KL(IBRPAR)
      KBR(1)=IX
      KBL(2)=IX
      KBR(2)=IX+1
      IF((IX+1).EQ.KR(IBRPAR))5150,5160
 5150 NOLFT=2
      GO TO 6000
 5160 KBL(3)=IX+1
      KBR(3)=KR(IBRPAR)
      NOLFT=3
      GO TO 6000
 5140 KBL(1)=IX
      KBR(1)=IX+1
      KBL(2)=IX+1
      KBR(2)=KR(IBRPAR)
      NOLFT=2
      GO TO 6000
 5130 IF((J.EQ.KL(IBRPAR)).OR.(J.EQ.KR(IBRPAR)))CALL ERR(10)
```

```
      KBL(1)=KL(IBRPAR)
      KBR(1)=J
      KBL(2)=J
      KBR(2)=KR(IBRPAR)
      NOLFT=2
 6000 IXX=STGPRB
      STGPRB=IXX
      STGPRB=STGPRB+1.
      GO TO 5000
C     *****DONE-- PRINT OUT THE RESULTS
 8000 PRINT 8010,(PROBNA(I),I=1,8)
 8010 FORMAT(1H1,8A10)
      PRINT 8020,BUB
 8020 FORMAT(1H0,31HOBJECTIVE FUNCTION AT OPTIMUM  ,G12.4)
      PRINT 8030
 8030 FORMAT(1H0,28HVARIABLE VALUES AT OPTIMUM--)
      CALL XPRINT(XBEST)
      END

      SUBROUTINE XPRINT(Z)
      COMMON/FIRST/KLO(100),KRO(100),KL(100),KR(100),XCODED(100),XBEST
     1(100),W(1100),CUTS(1100),ZLSTNO(700),ZLSTPA(700),LSTKL(700),
     2 LSTKR(700),ZLSLB(700),IBRVR(700),FLAG(700),KBL(3),KBR(3),
     3VARNAM(100),PROBNA(8),MAXVAR,MAXCUT,STMAX,MAXROW,MAXA,
     4 NMROWS,NUMVAR,ICHK(100),VAL,LFLG
      DIMENSION Z(1)
      PRINT 10
 10   FORMAT(1H0)
      IL=1
 20   IU=NUMVAR
      IF((NUMVAR-IL).GT.7)IU=IL+7
      PRINT 40,(VARNAM(I),I=IL,IU)
 40   FORMAT(1H0,3X,A5,7(12X,A5))
      IZ=IU-IL+1
      PRINT 45,(Z(I),I=IL,IU)
 45   FORMAT(1H ,G12.4,7(5X,G12.4))
      IL=IL+8
      IF(IL.LE.NUMVAR)GO TO 20
      PRINT 10
      RETURN
      END

      SUBROUTINE LPPR(Y)
      COMMON/FIRST/KLO(100),KRO(100),KL(100),KR(100),XCODED(100),XBEST
     1(100),W(1100),CUTS(1100),ZLSTNO(700),ZLSTPA(700),LSTKL(700),
     2 LSTKR(700),ZLSLB(700),IBRVR(700),FLAG(700),KBL(3),KBR(3),
     3VARNAM(100),PROBNA(8),MAXVAR,MAXCUT,STMAX,MAXROW,MAXA,
     4 NMROWS,NUMVAR,ICHK(100),VAL,LFLG
      DIMENSION Y(1)
      PRINT 10
 10   FORMAT(1H0,29HPACKED LP SOLUTION, I     X(I))
      IW=KRO(NUMVAR)
      DO 30 I=1,IW
      IF(ABS(Y(I)).GE.1.E-10)PRINT 20,I,Y(I)
 20   FORMAT(1H ,15X,I6,2X,G10.4)
 30   CONTINUE
      PRINT 40
 40   FORMAT(1H0)
      RETURN
      END
```

```
      SUBROUTINE ERR(I)
      PRINT100
  100 FORMAT(*1PROGRAM MOGG ABORTED BECAUSE....*)
      GOTO(101,102,103,104,105,106,107,108,109,110,111),I
  101 PRINT 201
      CALL EXIT
  102 PRINT 202
      CALL EXIT
  103 PRINT 203
      CALL EXIT
  104 PRINT 204
      CALL EXIT
  105 PRINT 205
      CALL EXIT
  106 PRINT 206
      CALL EXIT
  107 PRINT 207
      CALL EXIT
  108 PRINT 208
      CALL EXIT
  109 PRINT 209
      CALL EXIT
  110 PRINT 210
      CALL EXIT
  111 PRINT 211
      CALL EXIT
      RETURN
  201 FORMAT(*0VARIABLE CARDS OUT OF ORDER--LOOK NEAR MOGG LABEL 105*)
  202 FORMAT(*0MAXCUTS EXCEEDED--LOOK NEAR MOGG LABEL 117 OR 124*)
  203 FORMAT(*0MATRIX A EXCEEDED--LOOK NEAR MOGG LABEL 9400*)
  204 FORMAT(*0LPMAX EXCEEDED--LOOK NEAR MOGG LABEL 1005*)
  205 FORMAT(*0INITIAL LP INFEASIBLE--LOOK NEAR MOGG LABEL 1008*)
  206 FORMAT(*0LIST LENGTH EXCEEDED--LOOK NEAR MOGG LABEL 1020*)
  207 FORMAT(*0XCODED VIOLATES CUTS--LOOK NEAR MOGG LABEL 1060 OR 7025*)
  208 FORMAT(*0NO BRANCHING NODE FOUND--LOOK NEAR MOGG LABEL 5060*)
  209 FORMAT(*0NO FEASIBLE POINT FOUND--LOOK NEAR MOGG LABEL 5064*)
  210 FORMAT(*0NO BRANCHING POSSIBLE ON VARIABLE CHOSEN--LOOK NEAR MOGG
     1LABEL 5130*)
  211 FORMAT(*0FLAG COMPUTED IMPROPERLY--LOOK NEAR MOGG LABEL 5110*)
      END
```

```
      SUBROUTINE SCAIL
      COMMON/FIRST/KLO(100),KRO(100),KL(100),KR(100),XCODED(100),XBEST
     1(100),W(1100),CUTS(1100),ZLSTNO(700),ZLSTPA(700), LSTKL(700),
     2 LSTKR(700),ZLSLB(700),IBRVR(700),FLAG(700),KBL(3),KBR(3),
     3VARNAM(100),PROBNA(8),MAXVAR,MAXCUT, STMAX,MAXROW,MAXA,
     4 NMROWS,NUMVAR,ICHK(100),VAL,LFLG
      COMMON/WORK1/ B(350),X(350),Y(350),YTEMP(350),A(5000),E(5000),
     1  IA(5000),IE(5000),LA(1302),LE(2002),ICNAM(1302,2),KINBAS(1302),
     2  JH(350),ISTYPE(350),NAME(20),NTEMP(20),CMIN,COND,ERMAX,IFFEZ,
     3  INVFRQ,IOBJ,IROWP,ITCH,ITCHA,ITCNT,ITRFRQ,IVIN,IVOUT,JCOLP,KINP,
     4  XSTAT,NROW,NCOL,NELEM,NETA,NLELEM,NLETA,NOFLEM,NGETA,NUELEM,
     5  NUETA,SUMINF,K3
      COMMON/BLOCK/ ZTOLZE,ZTOLPV,ZTCOST,NOMAX,NTMAX,NEMAX,QRO,QMA,QBA,
     1  QF,QEO,QBL,QPL,QMI,QA,QB,QC,QE,QF,QG,QH,QT,QL,QM,QN,QO,QR,QU,QZ
      DO 100 IXX=2,NMROWS
      SMALL=1.E70
      BIG=-1.E70
      LAST=LA(NCOL+1)-1
      IFIRST=LA(NROW+1)
      DO 200 IXY=IFIRST,LAST
      IF(IA(IXY).NE.IXX)GOTO200
      IF(ABS(A(IXY)).LT.SMALL)SMALL=ABS(A(IXY))
      IF(ABS(A(IXY)).GT.BIG)BIG=ABS(A(IXY))
200   CONTINUE
      AV=SQRT(SMALL*BIG)
      ZL2AV=ALOG(AV)/ALOG(2.)
      L2AV=INT(ZL2AV)
      IF(ZL2AV.LT.0..AND.L2AV-ZL2AV.GE..5)L2AV=L2AV-1
      IF(ZL2AV.GT.0..AND.ZL2AV-L2AV.GE..5)L2AV=L2AV+1
      DIV=2.**L2AV
      DO 300 IXY=IFIRST,LAST
      IF(IA(IXY).NE.IXX)GOTO300
      A(IXY)=A(IXY)/DIV
300   CONTINUE
      B(IXX)=B(IXX)/DIV
100   CONTINUE
      RETURN
      END
```

```
      SUBROUTINE LINPRG
C
      COMMON/FIRST/KLO(100),KRO(100),KL(100),KR(100),XCODED(100),XBEST
     1(100),W(1100),CUTS(1100),ZLSTNO(700),ZLSTPA(700),LSTKL(700),
     2 LSTKR(700),ZLSLB(700),IBRVR(700),FLAG(700),KBL(3),KBR(3),
     3VARNAM(100),PROBNA(8),MAXVAR,MAXCUT,STMAX,MAXROW,MAXA,
     4 NMROWS,NUMVAR,ICHK(100),VAL,LFLG
      COMMON/WORK1/ B(350),X(350),Y(350),YTEMP(350),A(5000),E(5000),
     1  IA(5000),IE(5000),LA(1302),LE(2002),ICNAM(1302,2),KINBAS(1302),
     2  JH(350),ISTYPE(350),NAME(20),NTEMP(20),CMIN,COND,ERMAX,IFFEZ,
     3  INVFRQ,IOBJ,IROWP,ITCH,ITCHA,ITCNT,ITRFRQ,IVIN,IVOUT,JCOLP,KINP,
     4  XSTAT,NROW,NCOL,NELEM,NETA,NLELEM,NLETA,NGELEM,NGETA,NUELEM,
     5  NUETA,SUMINF,K3
      COMMON/BLOCK/ ZTOLZE,ZTOLPV,ZTCOST,NRMAX,NTMAX,NEMAX,QRO,QMA,QBA,
     1  QFI,QEO,QBL,QPL,QMI,QA,QB,QC,QE,QF,QG,QH,QT,QL,QM,QN,QO,QR,QU,QZ
C
C
C
      ITCNT=0
      ITCH=0
C        SET UP STARTING BASIS
      DO 9100 J=1,NCOL
 9100 KINBAS(J)=0
      DO 900 I=1,NROW
      ICOL=JH(I)
      ML=NROW
           DO 300 K=1,NUMVAR
           MR=KL(K)+NROW
           IF((ICOL.GT.ML).AND.(ICOL.LT.MR))GO TO 700
           ML=KR(K)+NROW
  300      CONTINUE
      IF(ICOL.GT.ML)GO TO 700
      GO TO 900
  700 JH(I)=I
  900 CONTINUE
C
C
 1000 CALL INVERT
      ITSINV = 0
      CALL ITEROP(0)
C
C                    SIMPLEX CYCLE
C
 1500 CALL FORMC
      CALL SHIFTR(3,4)
      ITCH=0
 1700 CALL BTRAN
      CALL PRICE
      IF (CMIN .LE.-ZTCOST) GO TO 3000
      IF (XSTAT .EQ. QI   ) GO TO 2000
      XSTAT = QBL
      GO TO 6000
 2000 XSTAT = QN
      GO TO 6000
 3000 CALL UNPACK(JCOLP)
      CALL FTRAN(1)
      ERMAX=0.
```

```
      DO 8000 I=1,NROW
      ERMAX=ERMAX+Y(I)*YTEMP(I)
 8000 CONTINUE
      DIFXX=CMIN-ERMAX
      DIFXX=ABS(DIFXX)
      IF (DIFXX.LE.ZTCOST)  GO TO 8500
      IF (K3.NE.1)  GO TO 8100
      PRINT 9500,CMIN,ERMAX
 9500 FORMAT(1H ,10X,6HCMIN= ,F16.8,5X,7HERMAX= ,F16.8)
 8100 IF (ERMAX.LE.0.)  GO TO 8500
      IF (ITCH.GT.0)  GO TO 1000
      ITCH=JCOLP
      ITCHA=ITCHA+1
      CALL SHIFTR(4,3)
      GO TO 1700
 8500 CONTINUE
      CALL CHUZR
      IF(XSTAT.EQ.QU)GO TO 6000
      IVOUT=JH(IROWP)
      IVIN = JCOLP
      CALL UPBETA
      KINBAS(JCOLP) = IROWP
      KINBAS(IVOUT) = 0
      JH(IROWP) = IVIN
      ITCNT = ITCNT + 1
      ITSINV = ITSINV + 1
      CALL ITEROP(1)
      IF (NELEM .GT. 5000) GO TO 1000
      CALL WRETA
      IF (ITSINV .GE. INVFRQ) GO TO 1000
      IF (ITCNT .GE. ITRFRQ) GO TO 6000
      GO TO 1500
C
 6000 CALL ITEROP(1)
C
C         SET PARMS
C
      DO 7000 I=1,NROW
      JHX=JH(I)
      IF (JHX.LE.NROW)  GO TO 6500
       W(JHX-NROW)=X(I)
 6500 CONTINUE
 7000 CONTINUE
C
C
      VAL=-X(IOBJ)
      LFLG=1
      IF (XSTAT.EQ.QBL)  LFLG=0
      PRINT 9600,ITCHA
 9600 FORMAT(1H+,108X,18HSTABILITY COUNT = ,I5)
 7100 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE FORMC
C
      COMMON/WORK1/ B(350),X(350),Y(350),YTEMP(350),A(5000),E(5000),
     1  IA(5000),IE(5000),LA(1302),LE(2002),ICNAM(1302,2),KINBAS(1302),
     2  JH(350),ISTYPE(350),NAME(20),NTEMP(20),CMIN,COND,ERMAX,IFFEZ,
     3  INVFRQ,IOBJ,IROWP,ITCH,ITCHA,ITCNT,ITRFRQ,IVIN,IVOUT,JCOLP,KINP,
     4  XSTAT,NROW,NCOL,NELEM,NETA,NLELEM,NLETA,NUFLEM,NGETA,NUELEM,
     5  NUETA,SUMINF,K3
      COMMON/BLOCK/ ZTOLZE,ZTOLPV,ZTCOST,NRMAX,NTMAX,NEMAX,QRO,QMA,QBA,
     1  QFI,QEO,QBL,QPL,QMI,QA,QB,QC,QE,QF,QG,QH,QI,QL,QM,QN,QO,QR,QU,QZ
C
C
      XSTAT=QF
      IFFEZ = 1
      DO 100 I = 1,NROW
      Y(I) = 0.
  100 CONTINUE
      SUM = 0.
C
      DO 1000 I = 1,NROW
      ICOL = JH(I)
      IF (ICOL .GT. NROW) GO TO 500
      IF (ISTYPE(ICOL)) 200,1000,500
C
  200 IF ( ABS(X(I)) .LE. ZTOLZE) GO TO 1000
      IF(X(I) .LT. 0.) Y(I) = +1.
      IF(X(I) .GT. 0.) Y(I) = -1.
      SUM = SUM +  ABS(X(I))
      GO TO 510
C
  500 IF(X(I) .GT. -ZTOLZE) GO TO 1000
      Y(I) = +1.
      SUM = SUM - X(I)
  510 IFFEZ = 0
      XSTAT = QI
 1000 CONTINUE
C
      SUMINF = SUM
      IF (IFFEZ .LE. 0) GO TO 9000
      Y(IOBJ) = 1.
C
 9000 RETURN
      END
```

```
      SUBROUTINE BTRAN
C
      COMMON/WORK1/ B(350),X(350),Y(350),YTEMP(350),A(5000),E(5000),
     1  IA(5000),IE(5000),LA(1302),LE(2002),ICNAM(1302,2),KINBAS(1302),
     2  JH(350),ISTYPE(350),NAME(20),NTEMP(20),CMIN,COND,ERMAX,IFFEZ,
     3  INVFRQ,IOBJ,IROWP,ITCH,ITCHA,ITCNT,ITRFRQ,IVIN,IVOUT,JCOLP,KINP,
     4  XSTAT,NROW,NCOL,NELEM,NETA,NLELEM,NLETA,NGELEM,NGETA,NUELEM,
     5  NUETA,SUMINF,K3
      COMMON/BLOCK/ ZTOLZE,ZTOLPV,ZTCOST,NRMAX,NTMAX,NEMAX,QRO,QMA,QBA,
     1  QFI,QEO,QBL,QPL,QMI,QA,QB,QC,QE,QF,QG,QH,QT,QL,QM,QN,QO,QR,QU,QZ
C
      IF (NETA .LE. 0) GO TO 9000
      DO 1000 I = 1,NETA
      IK = NETA - I + 1
      LL = LE(IK)
      KK = LE(IK+1) - 1
      IPIV = IE(LL)
      DP = E(LL)
      DY = Y(IPIV)
      DSUM = 0.
      IF (KK .LE. LL) GO TO 600
      LL = LL + 1
      DO 500 J = LL,KK
      IR = IE(J)
      DE = E(J)
      DPROD = DE * Y(IR)
      DSUM = DSUM + DPROD
  500 CONTINUE
C
  600 Y(IPIV) = (DY - DSUM) / DP
 1000 CONTINUE
C
 9000 RETURN
      END
```

```
      SUBROUTINE PRICE
C
      COMMON/FIRST/KLO(100),KRO(100),KL(100),KR(100),XCODED(100),XREST
     1(100),W(1100),CUTS(1100),ZLSTNO(700),ZLSTPA(700), LSTKL(700),
     2 LSTKR(700),ZLSLB(700),IBRVR(700),FLAG(700),KBL(3),KBR(3),
     3VARNAM(100),PROBNA(8),MAXVAR,MAXCUT, STMAX,MAXROW,MAXA,
     4 NMROWS,NUMVAR,ICHK(100),VAL,LFLG
      COMMON/WORK1/ B(350),X(350),Y(350),YTEMP(350),A(5000),E(5000),
     1  IA(5000),IE(5000),LA(1302),LE(2002),ICNAM(1302,2),KINBAS(1302),
     2  JH(350),ISTYPE(350),NAME(20),NTEMP(20),CMIN,COND,ERMAX,IFFEZ,
     3  INVFRQ,IOBJ,IROWP,ITCH,ITCHA,ITCNT,ITRFRQ,IVIN,IVOUT,JCOLP,KINP,
     4  XSTAT,NROW,NCOL,NELEM,NETA,NLELEM,NLETA,NUFLEM,NGETA,NUELEM,
     5  NUETA,SUMINF,K3
      COMMON/BLOCK/ ZTOLZE,ZTOLPV,ZTCOST,NRMAX,NTMAX,NEMAX,QRO,QMA,QBA,
     1  QFT,QEO,QBL,QPL,QMI,QA,QB,QC,QE,QF,QG,QH,QT,QL,QM,QN,QO,QR,QU,QZ
C
      JCOLP = 0
      CMIN = 1.E10
      DO 1000 J = 1,NCOL
      IF(J .LE. NROW .AND. ISTYPE(J) .NE. 1) GO TO 1000
      IF (KINBAS(J) .NE. 0) GO TO 1000
      IF (ITCH.EQ.J)  GO TO 1000
C
      QL=NROW
      DO 300 K=1,NUMVAR
      QR=KL(K)+NROW
      IF ((J.GT.QL) .AND. (J.LT.QR))  GO TO 1000
      QL=KR(K)+NROW
  300 CONTINUE
C
      IF (J.GT.QL)  GO TO 1000
      DSUM = 0.
      LL = LA(J)
      KK = LA(J+1) - 1
      DO 500 I = LL,KK
      IR = IA(I)
      DE = A(I)
      DPROD = DE * Y(IR)
      DSUM = DSUM + DPROD
  500 CONTINUE
      IF (DSUM .GE. CMIN) GO TO 1000
      CMIN = DSUM
      JCOLP = J
 1000 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE SHIFTR(IOLD,INEW)
C
      COMMON/WORK1/ B(350),X(350),Y(350),YTEMP(350),A(5000),E(5000),
     1  IA(5000),IE(5000),LA(1302),LE(2002),ICNAM(1302,2),KINBAS(1302),
     2  JH(350),ISTYPE(350),NAME(20),NTEMP(20),CMIN,COND,ERMAX,IFFEZ,
     3  INVFRQ,IOBJ,IROWP,ITCH,ITCHA,ITCNT,ITRFRQ,IVIN,IVOUT,JCOLP,KINP,
     4  XSTAT,NROW,NCOL,NELEM,NETA,NLELEM,NLETA,NUFLEM,NGETA,NUELEM,
     5  NUETA,SUMINF,K3
      COMMON/BLOCK/ ZTOLZE,ZTOLPV,ZTCOST,NRMAX,NTMAX,NEMAX,QRO,QMA,QBA,
     1  QFI,QEO,QBL,QPL,QMI,QA,QB,QC,QE,QF,QG,QH,QT,QL,QM,QN,QO,QR,QU,QZ
C
      DIMENSION BARRAY(1400)
      EQUIVALENCE (BARRAY(1),B(1))
C
      IFO = (IOLD - 1) * NRMAX
      IFN = (INEW- 1) * NRMAX
C
      DO 1000 I = 1,NROW
      BARRAY(IFN + I) = BARRAY(IFO + I)
 1000 CONTINUE
      RETURN
      END

      SUBROUTINE UNPACK(IV)
C
      COMMON/WORK1/ B(350),X(350),Y(350),YTEMP(350),A(5000),E(5000),
     1  IA(5000),IE(5000),LA(1302),LE(2002),ICNAM(1302,2),KINBAS(1302),
     2  JH(350),ISTYPE(350),NAME(20),NTEMP(20),CMIN,COND,ERMAX,IFFEZ,
     3  INVFRQ,IOBJ,IROWP,ITCH,ITCHA,ITCNT,ITRFRQ,IVIN,IVOUT,JCOLP,KINP,
     4  XSTAT,NROW,NCOL,NELEM,NETA,NLELEM,NLETA,NUFLEM,NGETA,NUELEM,
     5  NUETA,SUMINF,K3
      COMMON/BLOCK/ ZTOLZE,ZTOLPV,ZTCOST,NRMAX,NTMAX,NEMAX,QRO,QMA,QBA,
     1  QFI,QEO,QBL,QPL,QMI,QA,QB,QC,QE,QF,QG,QH,QT,QL,QM,QN,QO,QR,QU,QZ
C
      DO 100 I = 1,NROW
      Y(I) = 0.
  100 CONTINUE
C
      LL = LA(IV)
      KK = LA(IV+1) - 1
      DO 200 I = LL,KK
      IR = IA(I)
      Y(IR) = A(I)
  200 CONTINUE
C
      RETURN
      END
```

```
      SUBROUTINE FTRAN(IPAR)
C
      COMMON/WORK1/ B(350),X(350),Y(350),YTEMP(350),A(5000),E(5000),
     1  IA(5000),IE(5000),LA(1302),LE(2002),ICNAM(1302,2),KINBAS(1302),
     2  JH(350),ISTYPE(350),NAME(20),NTEMP(20),CMIN,COND,ERMAX,IFFEZ,
     3  INVFRQ,IOBJ,IROWP,ITCH,ITCHA,ITCNT,ITRFRQ,VIN,IVOUT,JCOLP,KINP,
     4  XSTAT,NROW,NCOL,NELEM,NETA,NLELEM,NLETA,NUELEM,NGETA,NUELEM,
     5  NUETA,SUMINF,K3
      COMMON/BLOCK/ ZTOLZE,ZTOLPV,ZTCOST,NRMAX,NTMAX,NEMAX,QRO,QMA,QBA,
     1  QFI,QEO,QBL,QPL,QMI,QA,QB,QC,QE,QF,QG,QH,QI,QL,QM,QN,QO,QR,QU,QZ
C
      GO TO (100,110),IPAR
  100 NFE = 1
      NLE = NETA
      GO TO 200
  110 NFE = NLETA + 1
      NLE = NETA
  200 IF (NFE .GT. NLE) GO TO 9000
      DO 1000 IK = NFE,NLE
      LL = LE(IK)
      KK = LE(IK+1)  - 1
      IPIV = IE(LL)
      DY = Y(IPIV)
      DY = DY/E(LL)
      Y(IPIV) = DY
      IF (KK .LE. LL) GO TO 1000
      LL = LL + 1
      DO 500 J = LL,KK
      IR = IE(J)
      Y(IR) = Y(IR) - E(J) * DY
  500 CONTINUE
 1000 CONTINUE
 9000 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE CHUZR
C
      COMMON/WORK1/ B(350),X(350),Y(350),YTEMP(350),A(5000),E(5000),
     1  IA(5000),IE(5000),LA(1302),LE(2002),ICNAM(1302,2),KINBAS(1302),
     2  JH(350),ISTYPE(350),NAME(20),NTEMP(20),CMIN,COND,ERMAX,IFFEZ,
     3  INVFRQ,IOBJ,IROWP,ITCH,ITCHA,ITCNT,ITRFRQ,IVIN,IVOUT,JCOLP,KINP,
     4  XSTAT,NROW,NCOL,NELEM,NETA,NLELEM,NLETA,NUELEM,NGETA,NUELEM,
     5  NUETA,SUMINF,K3
      COMMON/BLOCK/ ZTOLZE,ZIOLPV,ZTCOST,NQMAX,NTMAX,NEMAX,QRO,QMA,QBA,
     1  QFI,QEO,QBL,QPL,QMI,QA,QB,QC,QE,QF,QG,QH,QI,QL,QM,QN,QO,QR,QU,QZ
C
C          SELECT PIVOT ROW/VARIABLE TO LEAVE THE BASIS
C
      ZTOLCR=1.E-4
      ZTOLXX=1.E-10
      XMIN1=1.E10
      XMIN2=1.E10
      XMIN3=1.E10
      IROWP1=0
      IROWP2=0
      IROWP3=0
C
      DO 2000 I=1,NROW
      IF (ISTYPE(I).EQ.0)  GO TO 2000
      IF (ABS(Y(I)).LT.ZTOLCR)  GO TO 2000
      ICOL=JH(I)
      IF ((ICOL.LE.NROW).AND.(ISTYPE(I).LT.0))  GO TO 1000
      XRATIO=X(I)/Y(I)
      IF(XRATIO.LT.-ZTOLZE)GOTO2000
      IF(Y(I).LT.0.)GOTO2000
      IF (XRATIO.GT.XMIN1)  GO TO 2000
      XMIN1=XRATIO
      IROWP1=I
      GO TO 2000
C
 1000 IF (ABS(X(I)).LT.ZTOLZE)  GO TO 1500
      XRATIO=X(I)/Y(I)
      IF (XRATIO.LT.0.)  GO TO 2000
      IF (XRATIO.GT.XMIN2)  GO TO 2000
      XMIN2=XRATIO
      IROWP2=I
      GO TO 2000
C
 1500 XXX=ABS(Y(I))
      XRATIO=ZTOLXX/XXX
      IF (XRATIO.GT.XMIN3)  GO TO 2000
      XMIN3=XRATIO
      IROWP3=I
C
 2000 CONTINUE
C
C          TEST FOR OUTGOING VECTOR
C
      IROWP=IROWP1
      XRATIO=XMIN1
      IF (XRATIO.LE.XMIN2)  GO TO 3000
      IROWP=IROWP2
```

```
      XRATIO=XMIN2
C
 3000 IF (XRATIO.LE.XMIN3)  GO TO 4000
      IROWP=IROWP3
      XRATIO=XMIN3
C
 4000 IF (IROWP.LE.0)  XSTAT=QU
      I=IROWP
      IF (K3.NE.1)  RETURN
      PRINT 9000,IROWP,X(I),Y(I),JH(I)
 9000 FORMAT(8H IROWP= ,I4,2X,6HX(I)= ,F16.8,2X,6HY(I)= ,F16.8,2X
     1,7HJH(I)= ,I4)
      RETURN
      END

      SUBROUTINE UPBETA
C
      COMMON/WORK1/ B(350),X(350),Y(350),YTEMP(350),A(5000),E(5000),
     1  IA(5000),IE(5000),LA(1302),LE(2002),ICNAM(1302,2),KINBAS(1302),
     2  JH(350),ISTYPE(350),NAME(20),NTEMP(20),CMIN,COND,ERMAX,IFFEZ,
     3  INVFRQ,IOBJ,IROWP,ITCH,ITCHA,ITCNT,ITRFRQ,IVIN,IVOUT,JCOLP,KINP,
     4  XSTAT,NROW,NCOL,NELEM,NETA,NLELEM,NLETA,NUELEM,NGETA,NUELEM,
     5  NUETA,SUMINF,K3
      COMMON/BLOCK/ ZTOLZE,ZTOLPV,ZTCOST,NRMAX,NTMAX,NEMAX,QRO,QMA,QBA,
     1  QFI,QEO,QBL,QPL,QMI,QA,QB,QC,QE,QF,QG,QH,QI,QL,QM,QN,QO,QR,QU,QZ
C
      DE = X(IROWP)
      DP = DE/Y(IROWP)
      X(IROWP) = DP
      DO 1000 I = 1,NROW
      IF (I .EQ. IROWP) GO TO 1000
      DE = X(I)
      X(I) = DE - Y(I)*DP
 1000 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE WRETA
C
      COMMON/WORK1/ B(350),X(350),Y(350),YTEMP(350),A(5000),E(5000),
     1  IA(5000),IE(5000),LA(1302),LE(2002),ICNAM(1302,2),KINBAS(1302),
     2  JH(350),ISTYPE(350),NAME(20),NTEMP(20),CMIN,COND,ERMAX,IFFEZ,
     3  INVFRQ,IOBJ,IROWP,ITCH,ITCHA,ITCNT,ITRFRQ,IVIN,IVOUT,JCOLP,KINP,
     4  XSTAT,NROW,NCOL,NELEM,NETA,NLELEM,NLETA,NUFLEM,NGETA,NUELEM,
     5  NUETA,SUMINF,K3
      COMMON/BLOCK/ ZTOLZE,ZTOLPV,ZTCOST,NRMAX,NTMAX,NEMAX,QRO,QMA,QBA,
     1  QFI,QEO,QBL,QPL,QMI,QA,QB,QC,QE,QF,QG,QH,QT,QL,QM,QN,QO,QR,QU,QZ
C
      NELEM = NELEM + 1
      IE(NELEM) = IROWP
      E(NELEM) = Y(IROWP)
C
      DO 1000 I = 1,NROW
      IF (I .EQ. IROWP) GO TO 1000
      IF ( ABS(Y(I)) .LE. ZTOLZE) GO TO 1000
      NELEM = NELEM + 1
      IE(NELEM) = I
      E(NELEM) = Y(I)
 1000 CONTINUE
C
      NETA = NETA + 1
      LE(NETA+1) = NELEM + 1
      RETURN
      END

      SUBROUTINE ITEROP(IPAR)
C
      COMMON/WORK1/ B(350),X(350),Y(350),YTEMP(350),A(5000),E(5000),
     1  IA(5000),IE(5000),LA(1302),LE(2002),ICNAM(1302,2),KINBAS(1302),
     2  JH(350),ISTYPE(350),NAME(20),NTEMP(20),CMIN,COND,ERMAX,IFFEZ,
     3  INVFRQ,IOBJ,IROWP,ITCH,ITCHA,ITCNT,ITRFRQ,IVIN,IVOUT,JCOLP,KINP,
     4  XSTAT,NROW,NCOL,NELEM,NETA,NLELEM,NLETA,NUFLEM,NGETA,NUELEM,
     5  NUETA,SUMINF,K3
      COMMON/BLOCK/ ZTOLZE,ZTOLPV,ZTCOST,NRMAX,NTMAX,NEMAX,QRO,QMA,QBA,
     1  QFI,QEO,QBL,QPL,QMI,QA,QB,QC,QE,QF,QG,QH,QT,QL,QM,QN,QO,QR,QU,QZ
C
      IF (IPAR .EQ.0) GO TO 1000
      OBJ =-X(IOBJ)
      IF (IFFEZ .EQ. 0) OBJ = SUMINF
C
      IF(K3.NE.1)RETURN
      WRITE(6,8000) ITCNT,XSTAT,OBJ,IVIN,IVOUT,CMIN,
     1NETA,NELEM,TIMER
 8000 FORMAT(1H ,I5,4X,A4,2X,F10.8,4X,I6,4X,I6,4X,F16.8,4X,I6,I8,
     1F8.2  )
      GO TO 9000
 1000 IF(K3.NE.1)RETURN
      WRITE(6,8100)
 8100 FORMAT(//8HOITCOUNT,2X6HSTATUS,4X9HOBJ VALUE,8X,5HVECIN,5X6HVECOUT
     1,11X,2HDJ,12X,4HNETA,3X,5HNELEM,4X,4HTIME )
 9000 RETURN
      END
```

```
      SUBROUTINE INVERT
C
      COMMON/WORK1/ B(350),X(350),Y(350),YTEMP(350),A(5000),E(5000),
     1  IA(5000),IE(5000),LA(1302),LE(2002),ICNAM(1302,2),KINBAS(1302),
     2  JH(350),ISTYPE(350),NAME(20),NTEMP(20),CMIN,CONO,ERMAX,IFFEZ,
     3  INVFRQ,IOBJ,IROWP,ITCH,ITCHA,ITCNT,ITRFRQ,TVIN,IVOUT,JCOLP,KINP,
     4  XSTAT,NROW,NCOL,NELEM,NETA,NLELEM,NLETA,NGELEM,NGETA,NUELEM,
     5  NUETA,SUMINF,K3
      COMMON/BLOCK/ ZTOLZE,ZTOLPV,ZTCOST,NOMAX,NTMAX,NEMAX,QRO,QMA,QBA,
     1  QF,QEO,QBL,QPL,QMI,QA,QB,QC,QE,QF,QG,QH,QI,QL,QM,QN,QO,QR,QU,QZ
C
      INTEGER   MREG,MREG,VREG
      DIMENSION MREG(350),MREG(350),VREG(350)
C
C          SET PARAMETERS
C
      NETA = 0
      NLETA = 0
      NGETA = 0
      NUETA = 0
      NELEM = 0
      NLELEM = 0
      NGELEM = 0
      NUELEM = 0
      NABOVE = 0
      LE(1) = 1
      LR1 = 1
      KR1 = 0
      LR4 = NROW + 1
      KR4 = NROW
C
C          PUT SLACKS AND ARTIFICIALS IN PART 4 AND REST IN PART 1
C
      DO 100 I = 1,NROW
      IF (JH(I) .GT. NROW) GO TO 50
      LR4 = LR4 - 1
      MREG(LR4) = JH(I)
      VREG(LR4) = JH(I)
      GO TO 90
   50 KR1 = KR1 + 1
      VREG(KR1) = JH(I)
   90 MREG(I) = -1
      JH(I) = 0
  100 CONTINUE
C
      KR3 = LR4 -1
      LR3 = LR4
C
      DO 200 I = LR4,KR4
      IR = MREG(I)
      MREG(IR)= 0
      JH(IR) = IR
      KINBAS(IR) = IR
  200 CONTINUE
C
C                    PULL OUT VECTORS BELOW BUMP AND GET ROW COUNTS
C
```

```
      NBNONZ = KR4 - LR4 + 1
      IF (KR1 .EQ. 0) GO TO 1190
      J = LR1
  210 IV = VREG(J)
      LL = LA(IV)
      KK = LA(IV+1) -1
      IRCNT = 0
      DO 220 I = LL,KK
      NBNONZ = NBNONZ + 1
      IR = IA(I)
      IF (HREG(IR) .GE. 0) GO TO 220
      IRCNT = IRCNT + 1
      HREG(IR) = HREG(IR) - 1
      IRP = IR
  220 CONTINUE
      IF(IRCNT - 1) 230,250,300
  230 CONTINUE
      IF(K3.EQ.1)PRINT 8000
 8000 FORMAT(16H0MATRIX SINGULAR )
      KINBAS(IV) = 0
      VREG(J) = VREG(KR1)
      KR1 = KR1 - 1
      IF (J .GT. KR1) GO TO 310
      GO TO 210
C
  250 VREG(J) = VREG(KR1)
      KR1 = KR1 - 1
      LR3 = LR3 - 1
      VREG(LR3) = IV
      MREG(LR3) = IRP
      HREG(IRP) = 0
      JH(IRP) = IV
      KINBAS(IV) = IRP
      IF (J .GT. KR1) GO TO 310
      GO TO 210
  300 IF (J .GE. KR1) GO TO 310
      J = J+1
      GO TO 210
C
C                  PULL OUT REMAINING VECTORS ABOVE AND BELOW THE
C                  BUMP AND ESTABLISH MERIT COUNTS OF COLUMNS
C
  310 NVREM = 0
      IF(KR1 .EQ. 0) GO TO 1190
      J = IR1
  320 IV = VREG(J)
      LL = LA(IV)
      KK = LA(IV+1) - 1
      IRCNT = 0
      DO 800 I = LL,KK
      IR = IA(I)
      IF(HREG(IR) .NE. -2) GO TO 400
C
C         PIVOT ABOVE BUMP (PART OF L)
C
      NABOVE = NABOVE + 1
      IROWS = IR
      CALL UNPACK(IV)
```

```
      CALL WRETA
      NLETA = NETA
      JH(IR) = IV
      KINBAS(IV) = IR
      VREG(J) = VREG(KR1)
      KR1 = KR1 - 1
      NVREM = NVREM + 1
      HREG(IR) = IV
      GO TO 940
C
  400 IF (HREG(IR) .GE. 0) GO TO 800
      IRCNT = IRCNT + 1
      IRP = IR
  800 CONTINUE
C
      IF (IRCNT - 1) 810,900,1000
  810 CONTINUE
      IF(K3.EQ.1)PRINT 8000
      KINBAS(IV) = 0
      VREG(J) = VREG(KR1)
      NVREM = NVREM + 1
      KR1 = KR1 - 1
      IF (J .GT. KR1) GO TO 1010
      GO TO 320
C
C                     PUT VECTOR BELOW BUMP
C
  900 VREG(J) = VREG(KR1)
      NVREM = NVREM + 1
      KR1 = KR1 - 1
      LR3= LR3 - 1
      VREG(LR3) = IV
      MREG(LR3) = IRP
      HREG(IRP) = 0
      JH(IRP) = IV
      KINBAS(IV) = IRP
C
C                     CHANGE ROW COUNTS
C
  940 DO 950 II = LL,KK
      IIR = IA(II)
      IF (HREG(IIR) .GE. 0) GO TO 950
      HREG(IIR) = HREG(IIR) + 1
  950 CONTINUE
      IF (J .GT. KR1) GO TO 1010
      GO TO 320
 1000 IF (J .GE. KR1) GO TO 1010
      J = J+1
      GO TO 320
 1010 IF (NVREM .GT. 0)  GO TO 310
C
C                     GET MERIT COUNTS
C
 1020 IF (KR1 .EQ. 0)  GO TO 1190
      DO 1100 J = LR1,KR1
      IV = VREG(J)
      LL = LA(IV)
      KK = LA(IV+1) - 1
```

```
      IMCNT = 0
      DO 1050 I = LL,KK
      IR = IA(I)
      IF (HREG(IR) .GE. 0) GO TO 1050
      IMCNT = IMCNT - (HREG(IR) +1)
 1050 CONTINUE
      MREG(J) = IMCNT
 1100 CONTINUE
C
C                    SORT COLUMNS INTO MERIT ORDER
C                    USING SHELL SORT
C
      ISD = 1
 1106 IF (KR1 .LT. 2*ISD) GO TO 1108
      ISD = 2*ISD
      GO TO 1106
 1108 ISD = ISD - 1
C                    END OF INITIALIZATION
 1101 IF (ISD .LE. 0) GO TO 1107
      ISK = 1
 1102 ISJ = ISK
      ISL = ISK + ISD
      ISY = MREG(ISL)
      ISZ = VREG(ISL)
 1103 IF (ISY .LT. MREG(ISJ)) GO TO 1104
 1105 ISL = ISJ + ISD
      MREG(ISL) = ISY
      VREG(ISL) = ISZ
      ISK = ISK + 1
      IF ((ISK + ISD) .LE. KR1) GO TO 1102
      ISD = (ISD - 1) / 2
      GO TO 1101
 1104 ISL = ISJ + ISD
      MREG(ISL) = MREG(ISJ)
      VREG(ISL) = VREG(ISJ)
      ISJ = ISJ - ISD
      IF (ISJ .GT. 0) GO TO 1103
      GO TO 1105
 1107 CONTINUE
C                    END OF SORT ROUTINE
C
C                    PUT OUT BELOW BUMP ETAS (PART OF U)
C
 1190 NSLCK = 0
      NBELOW = 0
      NELAST = NEMAX
      NTLAST = NTMAX
      LE(NTLAST + 1) = NELAST + 1
C
      LR = LR3
      IF (LR3 .GE. LR4) LR = LR4
      IF (LR .GT. KR4) GO TO 2050
      JK = KR4 + 1
      DO 2000 JJ=LR,KR4
      JK = JK - 1
      IV = VREG(JK)
      I = MREG(JK)
      NBELOW = NBELOW + 1
```

```
      IF (IV .GT. NROW) GO TO 1200
      NSLCK = NSLCK + 1
 1200 LL = LA(IV)
      KK = LA(IV+1) -1
      IF (KK .GT. LL) GO TO 1300
 1250 IF (ABS(A(LL) - 1.) .LE. ZTOLZE) GO TO 2000
C
 1300 NUETA = NUETA + 1
      DO 1400 J = LL,KK
      IR = IA(J)
      IF (IR .EQ. I) GO TO 1390
      IE(NFLAST)=IR
      F(NELAST) = A(J)
      NELAST = NELAST - 1
      NUELEM = NUELEM + 1
      GO TO 1400
 1390 EP = A(J)
 1400 CONTINUE
      IE(NFLAST) = I
      F(NELAST) = EP
      LE(NTLAST) = NELAST
      NELAST = NELAST - 1
      NTLAST = NTLAST - 1
      NUELEM = NUELEM + 1
 2000 CONTINUE
 2050 IF(KR1 .EQ. 0) GO TO 3500
C
C                    DO L-U DECOMPOSITION OF BUMP
C
      DO 3000 J = LR1,KR1
      IV = VREG(J)
      CALL UNPACK(IV)
      CALL FTRAN(2)
      IROWP = 0
      IRCMIN = -999999
      DO 2100 I = 1,NROW
      IF ( ABS(Y(I)) .LE. ZTOLPV) GO TO 2100
      IF (HREG(I) .GE.0) GO TO 2100
      IF (HREG(I) .LE. IRCMIN) GO TO 2100
      IRCMIN = HREG(I)
      IROWP = I
 2100 CONTINUE
      IF (IROWP .GT. 0) GO TO 2150
      IF(K3.EQ.1)PRINT 8000
      KINBAS(IV) = 0
      GO TO 3000
C
 2150 INCR = HREG(IROWP) + 3
C
C                    WRITE L AND U ETAS
C
      IF (J .EQ. KR1) GO TO 2100
      NELEM = NELEM + 1
      IE(NFLEM) = IROWP
      F(NELEM) = Y(IROWP)
 2160 DO 2300 I = 1,NROW
      IF (I .EQ. IROWP) GO TO 2300
      IF( ABS(Y(I)) .LE. ZTOLZE) GO TO 2300
```

```
      IF (HREG(I) .GE. 0) GO TO 2200
C
C                     L ETA ELEMENTS
C
      NELEM = NELEM + 1
      IE(NELEM) = I
      F(NELEM) = Y(I)
      GO TO 2300
C
C                     U ETA ELEMENTS
C
 2200 IE(NELAST) = I
      F(NELAST) = Y(I)
      NELAST = NELAST - 1
      NUELEM = NUELEM + 1
 2300 CONTINUE
C
      JH(IROWP) = IV
      KINBAS(IV) = IROWP
      NUETA = NUETA + 1
      IE(NELAST) = IROWP
      IF (I .NE. KR1) GO TO 2330
      F(NELAST) = Y(IROWP)
      GO TO 2340
 2330 F(NELAST) = 1.
      NETA = NETA + 1
      LE(NETA+1) = NELEM + 1
 2340 NUELEM = NUELEM + 1
      LE(NTLAST) = NELAST
      NELAST = NELAST - 1
      NTLAST = NTLAST - 1
C
C                     UPDATE ROW COUNTS
C
      DO 2350 I = 1,NROW
      IF ( ABS(Y(I)) .LE. ZTOLZE) GO TO 2350
      IF (HREG(I) .GE. 0) GO TO 2350
      HREG(I) = HREG(I) - INCR
      IF (HREG(I) .GE. 0) HREG(I) = -1
 2350 CONTINUE
      HREG(IROWP) = 0
 3000 CONTINUE
C
C                     MERGE L AND U ETAS
C
 3500 NLETA = NETA
      NETA = NLETA + NUETA
      NLELEM = NELEM
      NELEM = NLELEM + NUELEM
      IF (NUELEM .EQ. 0) GO TO 3550
      CALL SHFTE
C
C                     INSERT SLACKS FOR DELETED COLUMNS
C
 3550 DO 3600 I = 1,NROW
      IF (JH(I) .NE. 0) GO TO 3600
      JH(I) = I
      IROWP = I
```

```
      CALL UNPACK(I)
      CALL FTRAN(1)
      CALL WRETA
 3600 CONTINUE
C
C                    UPDATE X
C
      CALL SHIFTR(1,3)
      CALL FTRAN(1)
      CALL SHIFTR(3,2)
C
C                    PRINT STATISTICS
C
      NOFD = NELEM - NETA
      NSTR = NROW - NSLCK
      IF(K3.NE.1)RETURN
      WRITE(6,500)NBNONZ,NSTR,NABOVE,NBELOW,NLELEM,NLETA,NUELEM,NUETA,
     1NOFD,NETA
  500 FORMAT(18H0INVERT STATISTICS/1H ,I4,14H NONZ IN BASIS/1H ,I4,
     128H STRUCTURAL COLUMNS IN BASIS/1H ,I4,19H VECTORS ABOVE BUMP/1H ,
     2I4,19H VECTORS BELOW BUMP/3H L1,I5,5H NONZ,I5,5H ETAS/3H U1,I5,
     35H NONZ,I5,5H ETAS/8H TOTALS:,I5,14H OFF DIAG NONZ,I5,5H ETAS )
C
      RETURN
      END

      SUBROUTINE SHFTE
C
      COMMON/WORK1/ B(350),X(350),Y(350),YTEMP(350),A(5000),E(5000),
     1  IA(5000),IE(5000),LA(1302),LE(2002),ICNAM(1302,2),KINBAS(1302),
     2  JH(350),ISTYPE(350),NAME(20),NTEMP(20),CMIN,COND,ERMAX,IFFEZ,
     3  INVFRQ,IOBJ,IROWP,ITCH,ITCHA,ITCNT,ITRFRQ,IVIN,IVOUT,JCOLP,KINP,
     4  XSTAT,NROW,NCOL,NELEM,NETA,NLELEM,NLETA,NUELEM,NGETA,NUELEM,
     5  NUETA,SUMINF,K3
      COMMON/BLOCK/ ZTOLZE,ZTOLPV,ZTCOST,NRMAX,NTMAX,NEMAX,QRO,QMA,QBA,
     1  QFI,QEO,QBL,QPL,QMI,QA,QB,QC,QE,QF,QG,QH,QI,QL,QM,QN,QO,QR,QU,QZ
C
C                    SHIFT IE AND E OF U ELEMENTS
C
      NF = NEMAX - NUELEM + 1
      INCR = 0
      DO 1000 I = NF,NEMAX
      INCR = INCR + 1
      IE(NLELEM + INCR) = IE(I)
      E(NLELEM + INCR) = E(I)
 1000 CONTINUE
C
      IDIF = NEMAX - NLELEM - NUELEM
      NF = NTMAX - NUETA + 1
      INCR = 0
      DO 2000 I = NF,NTMAX
      INCR = INCR + 1
      LE(NLETA + INCR) = LE(I) - IDIF
 2000 CONTINUE
      LE(NETA+1) = NELEM + 1
      RETURN
      END

      SUBROUTINE GETPHI(I,J,A,F)
      RETURN
      END
```